AA-DC93A-TE

OFFICE MENU

Application Programmer's Reference, Volume 2: Functions

digital™

AA-DC93A-TE
First Printing, February 1985

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

Copyright © 1985 by Digital Equipment Corporation. All rights reserved.

The postage prepaid Reader's Comments Form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

digital™
ALL-IN-1
DATATRIEVE
DEC
DECmail
DECmate
DECnet
DECspell
DECsystem-10
DECSYSTEM-20
DECtalk
DECUS
DECwriter
DIBOL
MASSBUS
PDP
P/OS
Professional
Rainbow
RSTS
RSX
UNIBUS
VAX
VMS
VT
Work Processor
WPS-PLUS

Contents

Volume 1: Flow Control

Preface

1 Overview

1.1 Introduction 1-1
1.2 ALL-IN-1 in the VAX Information Architecture 1-3
1.3 ALL-IN-1 Architecture and Basic Concepts 1-3
1.3.1 Form and Function 1-5
1.3.2 How ALL-IN-1 Is Organized 1-6

2 ALL-IN-1 Initialization and Flow Control

2.1 Introduction 2-1
2.2 ALL-IN-1 Initialization 2-2
2.2.1 Partial Initialization 2-3
2.3 ALL-IN-1 Office Menu Flow Control 2-3

3 Form Processing

3.1 Introduction 3-1
3.2 Form Libraries 3-2
3.2.1 Searching Form Libraries 3-3

3.3 The FORM and SHOW_FORM Functions 3-3
3.3.1 The FORM Function 3-3
3.3.2 The SHOW_FORM Function 3-4
3.4 ALL-IN-1 Named Data Conventions 3-4
3.4.1 ALL-IN-1 Form .TYPEs 3-5
3.4.2 The .MORE Directive 3-7
3.4.3 Searching for Named Data 3-7
3.4.4 Form Qualifiers 3-8
3.4.5 Common, or Shared, Form Qualifiers 3-10
3.5 The MENU Form 3-24
3.5.1 Menu Named Data 3-25
3.5.2 Menu Form Qualifiers 3-25
3.5.3 Processing Menus 3-28
3.5.4 Stacking Menu Options 3-32
3.5.5 Invoking Menu Options Indirectly 3-33
3.5.6 Creating and Processing Complex Menus 3-33
3.5.7 Menu Stacks and Menu Levels 3-36
3.6 The Entry Form 3-37
3.6.1 Entry Form Named Data 3-37
3.6.2 The Data Entry Process: Creating Files 3-38
3.6.3 The Data Entry Process: Accessing Data Files 3-44
3.6.4 Entry Form Qualifiers 3-44
3.6.5 Processing Entry Forms 3-54.1
3.7 The Argument Form 3-56
3.7.1 Argument Form Named Data 3-56
3.7.2 Argument Form Qualifiers 3-57
3.8 The Calc and Desk Forms 3-57
3.8.1 Calc Form Named Data 3-57
3.8.2 Calc Form Qualifiers 3-58
3.8.3 The Desk Calculator Form 3-58
3.9 The Search Form 3-60
3.9.1 Search Form Named Data 3-61
3.9.2 Search Expressions in the .TYPE Directive 3-63
3.9.3 Search Form Qualifiers 3-65
3.9.4 Searching for Strings: The .SFILE Directive 3-65
3.9.5 Formatting Record Collections: The .SOUT Directive 3-67
3.9.6 Search Form Processing 3-68
3.10 The Select Form 3-68
3.10.1 Select Form Named Data 3-69
3.10.2 Select Form Qualifiers 3-69
3.10.3 Selecting and Processing Records 3-71
3.11 EXIT SCREEN and the PRIOR Form 3-75

3.12 Dynamically Redefining Named Data 3-75
3.12.1 The Form .TYPE Parameter in FORM Function Calls 3-76
3.12.2 The /RESET Form Qualifier 3-77
3.12.3 Dynamically Redefining Entry Forms 3-78
3.13 Multipage Form Sets 3-78
3.14 Nonstandard Forms 3-80

4 Field Processing

4.1 Introduction 4-1
4.2 The FIELD Function 4-2
4.3 Defining Fields 4-2
4.4 Types of Field Qualifiers 4-4
4.5 /AUTO 4-5
4.5.1 Examples 4-5
4.6 /BLANK 4-5
4.6.1 Example 4-6
4.7 /CLEAR 4-6
4.7.1 Examples 4-6
4.8 /COMPUTE 4-7
4.8.1 Examples 4-7
4.9 /COPY 4-8
4.9.1 Examples 4-9
4.10 /DECIMAL 4-10
4.10.1 Examples 4-11
4.11 /DOC 4-11
4.11.1 Examples 4-12
4.12 /FILE_CHECK 4-14
4.12.1 Examples 4-15
4.13 /GET_SAVE 4-16
4.13.1 Examples 4-17
4.14 /HARD 4-18
4.14.1 Examples 4-18
4.15 /INVALID 4-19
4.15.1 Examples 4-22
4.16 /LIST 4-22
4.16.1 Examples 4-23
4.17 /OPTIONAL 4-23
4.17.1 Examples 4-24
4.18 /POST_FUNCTION 4-24
4.18.1 Examples 4-25
4.19 /PRE_FUNCTION 4-25
4.19.1 Examples 4-26

4.20 /PROMPT 4-26
4.20.1 Example 4-27
4.21 /PUT_SAVE 4-27
4.21.1 Examples 4-27
4.22 /RECOGNITION 4-28
4.22.1 Examples 4-30
4.23 /RSE_nnn 4-30
4.23.1 Examples 4-32
4.24 /SCROLL 4-33
4.24.1 /SCROLL with Indexed Files 4-35
4.24.2 SCROLL$FUNCTION 4-36
4.24.3 Example 4-40
4.24.4 Customized Scrolling 4-43
4.25 /SHOW 4-45
4.25.1 Examples 4-46
4.26 /STORE 4-46
4.27 /UNIQUE 4-47
4.27.1 Example 4-47
4.28 /USE_FORM 4-47
4.29 /VALID 4-47
4.29.1 Examples 4-50
4.30 Data Validation 4-51
4.30.1 Combining Validation Qualifiers 4-54
4.30.2 Controlling Record Collection Output 4-55
4.31 Default Keyboard Definitions 4-56
4.32 Defining the Keyboard 4-63
4.33 Terminator Processing 4-66
4.34 Field Processing 4-69

5 Generic Menu Design

5.1 The Primary Menu Form 5-1
5.1.1 Current Items 5-3
5.1.2 Select (SEL) 5-3
5.1.3 Create (C) 5-4
5.1.4 Edit (E) 5-5
5.1.5 Delete (D) 5-5
5.1.6 Print (P) 5-6
5.1.7 Read (R) 5-6
5.1.8 Index (I) 5-6
5.1.9 Option x (Ox) 5-6
5.1.10 Training (TR) 5-6
5.2 More Menu Options (M) 5-7
5.3 Secondary Menu Form - Additional Menu Options 5-7

Volume 2: Functions

Preface

6 Functions

6.1 Introduction 6-1
6.2 Function Call Syntax 6-2
6.3 Stacking Functions 6-4
6.3.1 Trapping Responses and Error Conditions 6-6
6.3.2 Function Loops 6-7
6.3.3 Forcing Displays 6-7
6.3.4 Other Features 6-7
6.4 Pipelining Functions 6-9
6.5 User-Defined Functions 6-10
6.6 Symbol Substitution in Functions 6-12
6.7 ACMSTASK 6-13
6.7.1 Syntax 6-13
6.7.2 Description 6-13
6.7.3 Function Parameters 6-13
6.7.4 Function Qualifiers 6-14
6.7.5 Examples 6-14
6.7.6 Special Features and Restrictions 6-15
6.8 ADE 6-16
6.8.1 Syntax 6-16
6.8.2 Description 6-16
6.8.3 Function Parameters 6-16
6.9 AI_CHECK 6-17
6.9.1 Syntax 6-17
6.9.2 Description 6-17
6.9.3 Function Parameters 6-17
6.9.4 Function Qualifiers 6-17
6.9.5 Examples 6-18
6.9.6 Special Features and Restrictions 6-18
6.10 APPEND 6-19
6.10.1 Syntax 6-19
6.10.2 Description 6-19
6.10.3 Function Parameters 6-19
6.10.4 Examples 6-20
6.11 ATTACH 6-21
6.11.1 Syntax 6-21

6.11.2 Description 6-21
6.11.3 Function Parameters 6-21
6.11.4 Function Qualifiers 6-21
6.11.5 Example 6-21
6.11.6 Special Features and Restrictions 6-22
6.12 BYE 6-23
6.12.1 Syntax 6-23
6.12.2 Description 6-23
6.12.3 Function Parameters 6-23
6.12.4 Function Qualifiers 6-23
6.12.5 Examples 6-24
6.12.6 Special Features and Restrictions 6-24
6.13 CABINET 6-25
6.13.1 Syntax 6-25
6.13.2 Description 6-25
6.13.3 Subfunction Descriptions 6-25
6.14 CALENDAR 6-26
6.14.1 Syntax 6-26
6.14.2 Description 6-26
6.14.3 Subfunction Descriptions 6-26
6.14A CHECK_DISKQUOTA 6-26.1
6.14A.1 Syntax 6-26.1
6.14A.2 Description 6-26.1
6.14A.3 Function Parameters 6-26.1
6.14A.4 Function Qualifiers 6-26.2
6.14A.5 Examples 6-26.2
6.14A.6 Special Features and Restrictions 6-26.2
6.15 CLEAR 6-27
6.15.1 Syntax 6-27
6.15.2 Description 6-27
6.15.3 Function Parameters 6-27
6.15.4 Function Qualifiers 6-27
6.15.5 Examples 6-28
6.15.6 Special Features and Restrictions 6-28
6.16 CLOSE_PRIOR 6-29
6.16.1 Syntax 6-29
6.16.2 Description 6-29
6.16.3 Function Parameters 6-29
6.16.4 Function Qualifiers 6-29
6.16.5 Examples 6-30
6.17 COMMAND 6-31
6.17.1 Syntax 6-31
6.17.2 Description 6-31
6.17.3 Function Parameters 6-31

6.17.4 Function Qualifiers 6-32
6.17.5 Examples 6-33
6.17.6 Special Features and Restrictions 6-34
6.18 COMPUTE 6-37
6.18.1 Syntax 6-37
6.18.2 Description 6-37
6.18.3 Function Parameters 6-37
6.18.4 Function Qualifiers 6-39
6.18.5 Examples 6-39
6.18.6 Special Features and Restrictions 6-41
6.19 CONTROL_C 6-42
6.19.1 Syntax 6-42
6.19.2 Description 6-42
6.19.3 Function Parameters 6-42
6.20 COPY 6-43
6.20.1 Syntax 6-43
6.20.2 Description 6-43
6.20.3 Function Parameters 6-43
6.20.4 Function Qualifiers 6-44
6.20.5 Examples 6-44
6.20.6 Special Features and Restrictions 6-44
6.21 CREATE 6-45
6.21.1 Syntax 6-45
6.21.2 Description 6-45
6.21.3 Function Parameters 6-45
6.21.4 Function Qualifiers 6-46
6.21.5 Examples 6-46
6.21.6 Special Features and Restrictions 6-46
6.22 CXP 6-47
6.22.1 Syntax 6-47
6.22.2 Description 6-47
6.22.3 Subfunction Descriptions 6-47
6.23 DATE_CONVERT 6-48
6.23.1 Syntax 6-48
6.23.2 Description 6-48
6.23.3 Function Parameters 6-48
6.23.4 Function Qualifiers 6-49
6.23.5 Examples 6-50
6.24 DCL 6-51
6.24.1 Syntax 6-51
6.24.2 Description 6-51
6.24.3 Function Parameters 6-51
6.24.4 Function Qualifiers 6-52
6.24.5 Examples 6-52
6.24.6 Special Features and Restrictions 6-52

6.25 DECIMAL 6-54
6.25.1 Syntax 6-54
6.25.2 Description 6-54
6.25.3 Function Parameters 6-54
6.25.4 Function Qualifiers 6-55
6.25.5 Examples 6-55
6.25.6 Special Features and Restrictions 6-55
6.26 DELETE_FILE 6-56
6.26.1 Syntax 6-56
6.26.2 Description 6-56
6.26.3 Function Parameters 6-56
6.26.4 Function Qualifiers 6-57
6.26.5 Examples 6-57
6.26.6 Special Features and Restrictions 6-57
6.27 DESTINATION 6-58
6.27.1 Syntax 6-58
6.27.2 Description 6-58
6.27.3 Function Parameters 6-58
6.27.4 Function Qualifiers 6-58
6.27.5 Special Features and Restrictions 6-58
6.28 DISPLAY 6-59
6.28.1 Syntax 6-59
6.28.2 Description 6-59
6.28.3 Function Parameters 6-59
6.28.4 Function Qualifiers 6-60
6.28.5 Examples 6-60
6.28.6 Special Features and Restrictions 6-60
6.29 DO 6-63
6.29.1 Syntax 6-63
6.29.2 Description 6-63
6.29.3 Function Parameters 6-63
6.29.4 Function Qualifiers 6-64
6.29.5 Examples 6-64
6.29.6 Special Features and Restrictions 6-64
6.30 DTR 6-65
6.30.1 Syntax 6-65
6.30.2 Description 6-65
6.30.3 Function Parameters 6-65
6.30.4 Function Qualifiers 6-66
6.30.5 Examples 6-66
6.30.6 Special Features and Restrictions 6-67

6.31 DUMP_CACHE 6-69
6.31.1 Syntax 6-69
6.31.2 Description 6-69
6.31.3 Function Parameters 6-69
6.31.4 Function Qualifiers 6-69
6.31.5 Examples 6-70
6.31.6 Special Features and Restrictions 6-70
6.32 EDIT 6-71
6.32.1 Syntax 6-71
6.32.2 Description 6-71
6.32.3 Function Parameters 6-71
6.32.4 Function Qualifiers 6-73
6.32.5 Special Features and Restrictions 6-73
6.33 EDT 6-74
6.33.1 Syntax 6-74
6.33.2 Description 6-74
6.33.3 Function Parameters 6-74
6.33.4 Function Qualifiers 6-74
6.33.5 Special Features and Restrictions 6-75
6.34 ERROR 6-76
6.34.1 Syntax 6-76
6.34.2 Description 6-76
6.34.3 Function Parameters 6-76
6.34.4 Function Qualifiers 6-76
6.34.5 Examples 6-77
6.34.6 Special Features and Restrictions 6-77
6.35 EXIT 6-78
6.35.1 Syntax 6-78
6.35.2 Description 6-78
6.35.3 Function Parameters 6-78
6.35.4 Function Qualifiers 6-78
6.35.5 Examples 6-79
6.35.6 Special Features and Restrictions 6-79
6.36 FHELP 6-80
6.36.1 Syntax 6-80
6.36.2 Description 6-80
6.36.3 Function Parameters 6-80
6.36.4 Function Qualifiers 6-80
6.36.5 Examples 6-80

6.37 FIELD 6-81
6.37.1 Syntax 6-81
6.37.2 Description 6-81
6.37.3 Function Parameters 6-81
6.37.4 Function Qualifiers 6-82
6.37.5 Examples 6-82
6.37.6 Special Features and Restrictions 6-82
6.38 FOR 6-83
6.38.1 Syntax 6-83
6.38.2 Description 6-83
6.38.3 Function Parameters 6-83
6.38.4 Boolean Expressions 6-85
6.38.5 Special DO Subfunctions 6-87
6.38.6 Examples 6-88
6.38.7 Special Features and Restrictions 6-90
6.39 FORCE 6-92
6.39.1 Syntax 6-92
6.39.2 Description 6-92
6.39.3 Function Parameters 6-92
6.39.4 Function Qualifiers 6-92
6.39.5 Examples 6-93
6.40 FORM 6-94
6.40.1 Syntax 6-94
6.40.2 Description 6-94
6.40.3 Function Parameters 6-94
6.40.4 Form Qualifiers 6-96
6.41 FORMAT 6-97
6.41.1 Syntax 6-97
6.41.2 Description 6-97
6.41.3 Function Parameters 6-97
6.41.4 Function Qualifiers 6-99
6.41.5 Examples 6-99
6.41.6 Special Features and Restrictions 6-100
6.42 GET 6-101
6.42.1 Syntax 6-101
6.42.2 Description 6-101
6.42.3 Function Parameters 6-102
6.42.4 Function Qualifiers 6-103
6.42.5 Examples 6-103
6.42.6 Special Features and Restrictions 6-107

6.43 HELP 6-108
6.43.1 Syntax 6-108
6.43.2 Description 6-108
6.43.3 Function Parameters 6-108
6.43.4 Function Qualifiers 6-109
6.43.5 Examples 6-110
6.43.6 Special Features and Restrictions 6-110
6.44 IFEXIT 6-112
6.44.1 Syntax 6-112
6.44.2 Description 6-112
6.44.3 Function Parameters 6-112
6.44.4 Function Qualifiers 6-112
6.44.5 Examples 6-113
6.44.6 Special Features and Restrictions 6-113
6.45 INCLUDE 6-114
6.45.1 Syntax 6-114
6.45.2 Description 6-114
6.45.3 Function Parameters 6-114
6.45.4 Function Qualifiers 6-114
6.45.5 Examples 6-115
6.45.6 Special Features and Restrictions 6-115
6.46 KEYPAD 6-116
6.46.1 Syntax 6-116
6.46.2 Description 6-116
6.46.3 Function Parameters 6-116
6.46.4 Function Qualifiers 6-116
6.46.5 Examples 6-117
6.46.6 Special Features and Restrictions 6-117
6.47 LIGHTS 6-118
6.47.1 Syntax 6-118
6.47.2 Description 6-118
6.47.3 Function Parameters 6-118
6.47.4 Examples 6-118
6.47.5 Special Features and Restrictions 6-119
6.48 LIST 6-120
6.48.1 Syntax 6-120
6.48.2 Description 6-120
6.48.3 Function Parameters 6-120
6.48.4 Function Qualifiers 6-121
6.48.5 Examples 6-121
6.48.6 Special Features and Restrictions 6-122

6.49 LOG 6-124
6.49.1 Syntax 6-124
6.49.2 Description 6-124
6.49.3 Function Parameters 6-124
6.49.4 Function Qualifiers 6-124
6.49.5 Examples 6-125
6.49.6 Special Features and Restrictions 6-125
6.50 LOGICAL 6-126
6.50.1 Syntax 6-126
6.50.2 Description 6-126
6.50.3 Function Parameters 6-126
6.50.4 Function Qualifiers 6-127
6.50.5 Examples 6-127
6.50.6 Special Features and Restrictions 6-128
6.51 MAIL 6-129
6.51.1 Syntax 6-129
6.51.2 Description 6-129
6.51.3 Subfunction Descriptions 6-129
6.52 MAKE_FILE_NAME 6-130
6.52.1 Syntax 6-130
6.52.2 Description 6-130
6.52.3 Function Parameters 6-130
6.52.4 Function Qualifiers 6-132
6.52.5 Examples 6-132
6.52.6 Special Features and Restrictions 6-133
6.53 MERGE 6-134
6.53.1 Syntax 6-134
6.53.2 Description 6-134
6.53.3 Function Parameters 6-134
6.53.4 Template Directives 6-137
6.53.5 Function Qualifiers 6-139
6.53.6 Examples 6-139
6.53.7 Special Features and Restrictions 6-140
6.54 MERGE_LINE 6-141
6.54.1 Syntax 6-141
6.54.2 Description 6-141
6.54.3 Function Parameters 6-141
6.54.4 Function Qualifiers 6-141
6.54.5 Examples 6-142
6.54.6 Special Features and Restrictions 6-142

6.55 MERGE_LIST 6-143
6.55.1 Syntax 6-143
6.55.2 Description 6-143
6.55.3 Function Parameters 6-143
6.55.4 Function Qualifiers 6-145
6.55.5 Examples 6-146
6.55.6 Special Features and Restrictions 6-146
6.56 MODE 6-147
6.56.1 Syntax 6-147
6.56.2 Description 6-147
6.56.3 Function Parameters 6-147
6.56.4 Function Qualifiers 6-148
6.56.5 Examples 6-148
6.56.6 Special Features and Restrictions 6-149
6.57 NEWDIR 6-150
6.57.1 Syntax 6-150
6.57.2 Description 6-150
6.57.3 Function Parameters 6-150
6.57.4 Function Qualifiers 6-151
6.57.5 Examples 6-151
6.57.6 Special Features and Restrictions 6-151
6.58 NEWLIB 6-152
6.58.1 Syntax 6-152
6.58.2 Description 6-152
6.58.3 Function Parameters 6-152
6.58.4 Function Qualifiers 6-153
6.58.5 Examples 6-153
6.58.6 Special Features and Restrictions 6-153
6.59 NEXT_LIST 6-155
6.59.1 Syntax 6-155
6.59.2 Description 6-155
6.59.3 Function Parameters 6-155
6.59.4 Function Qualifiers 6-155
6.59.5 Examples 6-156
6.59.6 Special Features and Restrictions 6-156
6.60 OA$CLI_GET_SYMBOL 6-157
6.60.1 Syntax 6-157
6.60.2 Description 6-157
6.60.3 Function Parameters 6-157
6.60.4 Function Qualifiers 6-157
6.60.5 Examples 6-158
6.60.6 Special Features and Restrictions 6-158

6.61 OA$CLI_GET_VALUE 6-159
6.61.1 Syntax 6-159
6.61.2 Description 6-159
6.61.3 Function Parameters 6-159
6.61.4 Function Qualifiers 6-159
6.61.5 Examples 6-160
6.61.6 Special Features and Restrictions 6-160
6.62 OA$CNV_CONVERT 6-161
6.62.1 Syntax 6-161
6.62.2 Description 6-161
6.62.3 Function Parameters 6-162
6.62.4 Function Qualifiers 6-162
6.62.5 Examples 6-162
6.62.6 Special Features and Restrictions 6-162
6.63 OA$CNV_DIST_LIST 6-163
6.63.1 Syntax 6-163
6.63.2 Description 6-163
6.63.3 Function Parameters 6-163
6.63.4 Examples 6-164
6.63.5 Special Features and Restrictions 6-164
6.64 OA$CNV_MERGE 6-165
6.64.1 Syntax 6-165
6.64.2 Description 6-165
6.64.3 Function Parameters 6-166
6.64.4 Function Qualifiers 6-166
6.64.5 Examples 6-166
6.64.6 Special Features and Restrictions 6-166
6.65 OA$CNV_FROM_SCROLL 6-167
6.65.1 Syntax 6-167
6.65.2 Description 6-167
6.65.3 Function Parameters 6-167
6.65.4 Function Qualifiers 6-167
6.66 OA$CNV_TO_SCROLL 6-168
6.66.1 Syntax 6-168
6.66.2 Description 6-168
6.66.3 Function Parameters 6-168
6.66.4 Function Qualifiers 6-168
6.67 OA$DSA_CACHE_STATUS 6-169
6.67.1 Syntax 6-169
6.67.2 Description 6-169
6.67.3 Function Parameters 6-169
6.67.4 Function Qualifiers 6-169
6.67.5 Examples 6-170
6.67.6 Special Features and Restrictions 6-170

6.68 OA$ENT_SET_xxx 6-171
6.68.1 Syntax 6-171
6.68.2 Description 6-171
6.68.3 Function Parameters 6-171
6.68.4 Function Qualifiers 6-171
6.68.5 Special Features and Restrictions 6-172
6.69 OA$FBT_INSTALL_LIBRARY 6-173
6.69.1 Syntax 6-173
6.69.2 Description 6-173
6.69.3 Function Parameters 6-173
6.69.4 Function Qualifiers 6-174
6.69.5 Examples 6-174
6.69.6 Special Features and Restrictions 6-174
6.70 OA$FBT_LIST_LIBTREE 6-175
6.70.1 Syntax 6-175
6.70.2 Description 6-175
6.70.3 Function Parameters 6-175
6.70.4 Function Qualifiers 6-175
6.70.5 Examples 6-176
6.71 OA$FBT_LIST_TREE 6-177
6.71.1 Syntax 6-177
6.71.2 Description 6-177
6.71.3 Function Parameters 6-177
6.71.4 Function Qualifiers 6-177
6.71.5 Examples 6-178
6.72 OA$FBT_REMOVE_LIBRARY 6-179
6.72.1 Syntax 6-179
6.72.2 Description 6-179
6.72.3 Function Parameters 6-179
6.72.4 Function Qualifiers 6-179
6.72.5 Examples 6-180
6.72.6 Special Features and Restrictions 6-180
6.73 OA_$FBT_WRITE_LIBRARY 6-181
6.73.1 Syntax 6-181
6.73.2 Description 6-181
6.73.3 Function Parameters 6-181
6.73.4 Function Qualifiers 6-182
6.73.5 Examples 6-182
6.73.6 Special Features and Restrictions 6-182

6.74 OA$FLD_nnn 6-183
6.74.1 Syntax 6-183
6.74.2 Description 6-183
6.74.3 Terminator Functions 6-183
6.74.4 Examples 6-186
6.75 OA$FLD_NEXT_LIST 6-188
6.75.1 Syntax 6-188
6.75.2 Description 6-188
6.75.3 Function Parameters 6-188
6.75.4 Function Qualifiers 6-188
6.76 OA$FLD_RECOGNITION 6-189
6.76.1 Syntax 6-189
6.76.2 Description 6-189
6.76.3 Function Parameters 6-189
6.76.4 Examples 6-189
6.76.5 Special Features and Restrictions 6-189
6.77 OA$FLD_SCROLL_OFF 6-190
6.77.1 Syntax 6-190
6.77.2 Description 6-190
6.77.3 Function Parameters 6-190
6.77.4 Examples 6-190
6.78 OA$FLD_SEARCH 6-191
6.78.1 Syntax 6-191
6.78.2 Description 6-191
6.78.3 Function Parameters 6-191
6.78.4 Examples 6-191
6.78.5 Special Features and Restrictions 6-191
6.79 OA$FLO_CLEAR_VA 6-192
6.79.1 Syntax 6-192
6.79.2 Description 6-192
6.79.3 Function Parameters 6-192
6.79.4 Function Qualifiers 6-192
6.79.5 Special Features and Restrictions 6-192
6.80 OA$FLO_CLOSE_LIB 6-193
6.80.1 Syntax 6-193
6.80.2 Description 6-193
6.80.3 Function Parameters 6-193
6.80.4 Function Qualifiers 6-193
6.80.5 Special Features and Restrictions 6-194

6.81 OA$FLO_OPEN_LIB 6-195
6.81.1 Syntax 6-195
6.81.2 Description 6-195
6.81.3 Function Parameters 6-195
6.81.4 Function Qualifiers 6-195
6.81.5 Special Features and Restrictions 6-196
6.82 OA$FRM_SET_FIELD 6-197
6.82.1 Syntax 6-197
6.82.2 Description 6-197
6.82.3 Function Parameters 6-197
6.82.4 Function Qualifiers 6-197
6.83 OA$HAR_CHECK_DECCRT 6-198
6.83.1 Syntax 6-198
6.83.2 Description 6-198
6.83.3 Function Parameters 6-198
6.83.4 Function Qualifiers 6-198
6.83.5 Special Features and Restrictions 6-199
6.84 This section has been removed 6-200
6.85 OA$INI_GLOBALS 6-201
6.85.1 Syntax 6-201
6.85.2 Description 6-201
6.85.3 Function Parameters 6-201
6.85.4 Function Qualifiers 6-201
6.86 OA$INI_INITIALIZE 6-202
6.86.1 Syntax 6-202
6.86.2 Description 6-202
6.86.3 Function Parameters 6-202
6.86.4 Function Qualifiers 6-202
6.86.5 Examples 6-203
6.87 OA$INI_LOGICALS 6-204
6.87.1 Syntax 6-204
6.87.2 Description 6-204
6.87.3 Function Parameters 6-204
6.87.4 Function Qualifiers 6-204
6.88 OA$INI_OPEN_LIB 6-205
6.88.1 Syntax 6-205
6.88.2 Description 6-205
6.88.3 Function Parameters 6-205
6.88.4 Function Qualifiers 6-205

6.89 OA$INI_SET_DEFAULT_DIR 6-206
6.89.1 Syntax 6-206
6.89.2 Description 6-206
6.89.3 Function Parameters 6-206
6.89.4 Function Qualifiers 6-206
6.90 OA$INI_SUBPROCESS 6-207
6.90.1 Syntax 6-207
6.90.2 Description 6-207
6.90.3 Function Parameters 6-207
6.90.4 Function Qualifiers 6-207
6.90.5 Examples 6-208
6.91 OA$KEY_COMMAND 6-209
6.91.1 Syntax 6-209
6.91.2 Description 6-209
6.91.3 Function Parameters 6-209
6.91.4 Special Features and Restrictions 6-209
6.92 OA$LIST_xxx 6-210
6.92.1 Syntax 6-210
6.92.2 Description 6-210
6.92.3 Function Parameters 6-210
6.92.4 Function Qualifiers 6-210
6.92.5 Examples 6-211
6.92.6 Special Features and Restrictions 6-211
6.93 OA$LOG_DISPLAY 6-212
6.93.1 Syntax 6-212
6.93.2 Description 6-212
6.93.3 Function Parameters 6-212
6.93.4 Function Qualifiers 6-212
6.93.5 Special Features and Restrictions 6-213
6.94 OA$LOG_TO_FILE 6-214
6.94.1 Syntax 6-214
6.94.2 Description 6-214
6.94.3 Function Parameters 6-214
6.94.4 Function Qualifiers 6-215
6.94.5 Examples 6-215
6.95 OA$MENU_LEVEL_PUSH 6-216
6.95.1 Syntax 6-216
6.95.2 Description 6-216
6.95.3 Function Parameters 6-217
6.95.4 Function Qualifiers 6-217

6.96 OA$MENU_SHOW_STACK 6-218
6.96.1 Syntax 6-218
6.96.2 Description 6-218
6.96.3 Function Parameters 6-219
6.96.4 Function Qualifiers 6-219
6.97 OA$MRG_TTOUT 6-220
6.97.1 Syntax 6-220
6.97.2 Description 6-220
6.97.3 Function Parameters 6-220
6.97.4 Examples 6-221
6.98 OA$MSG_PURGE 6-222
6.98.1 Syntax 6-222
6.98.2 Description 6-222
6.98.3 Function Parameters 6-222
6.98.4 Examples 6-222
6.99 OA$MSG_TRAP 6-223
6.99.1 Syntax 6-223
6.99.2 Description 6-223
6.99.3 Function Parameters 6-223
6.99.4 Function Qualifiers 6-223
6.100 OA$SCL_nnn 6-224
6.100.1 Syntax 6-224
6.100.2 Description 6-224
6.100.3 Scroll Functions 6-224
6.100.4 File Conversion for /SCROLL 6-227
6.100.5 Special Features and Restrictions 6-227
6.101 OA$SCP_STACK_DUMP 6-228
6.101.1 Syntax 6-228
6.101.2 Description 6-228
6.101.3 Function Parameters 6-228
6.101.4 Function Qualifiers 6-228
6.102 OA$STAT_GET_STATISTICS 6-229
6.102.1 Syntax 6-229
6.102.2 Description 6-229
6.102.3 Function Parameters 6-230
6.102.4 Function Qualifiers 6-230
6.102.5 Special Features and Restrictions 6-230
6.103 OA$SUB_CLOSE 6-231
6.103.1 Syntax 6-231
6.103.2 Description 6-231
6.103.3 Function Parameters 6-231
6.103.4 Function Qualifiers 6-231

6.104 OA$SUB_OPEN 6-232
6.104.1 Syntax 6-232
6.104.2 Description 6-232
6.104.3 Function Parameters 6-232
6.104.4 Function Qualifiers 6-232
6.105 OA$SYM_CLOSE 6-233
6.105.1 Syntax 6-233
6.105.2 Description 6-233
6.105.3 Function Parameters 6-233
6.105.4 Function Qualifiers 6-233
6.106 OA$SYM_GET_SYMBOL 6-234
6.106.1 Syntax 6-234
6.106.2 Description 6-234
6.106.3 Function Parameters 6-234
6.106.4 Function Qualifiers 6-235
6.106.5 Examples 6-235
6.107 OA$SYM_MAKE_SYMBOL 6-236
6.107.1 Syntax 6-236
6.107.2 Description 6-236
6.107.3 Function Parameters 6-236
6.107.4 Function Qualifiers 6-236
6.107.5 Examples 6-237
6.108 OA$SYM_OPEN 6-238
6.108.1 Syntax 6-238
6.108.2 Description 6-238
6.108.3 Function Parameters 6-238
6.108.4 Function Qualifiers 6-238
6.108.5 Special Features and Restrictions 6-239
6.109 OA$TRA_OUT 6-240
6.109.1 Syntax 6-240
6.109.2 Description 6-240
6.109.3 Function Parameters 6-240
6.109.4 Function Qualifiers 6-240
6.110 OA$TRA_SET 6-241
6.110.1 Syntax 6-241
6.110.2 Description 6-241
6.110.3 Function Parameters 6-241
6.110.4 Function Qualifiers 6-242
6.110.5 Examples 6-242
6.110.6 Special Features and Restrictions 6-243

6.111 OA$TRM_BRDCST_INIT 6-244
6.111.1 Syntax 6-244
6.111.2 Description 6-244
6.111.3 Function Parameters 6-244
6.111.4 Function Qualifiers 6-244
6.111.5 Examples 6-244
6.111A OA$TRM_BRDCST_STOP 6-244.2
6.111A.1 Syntax 6-244.2
6.111A.2 Description 6-244.2
6.111A.3 Function Parameters 6-244.2
6.111A.4 Function Qualifiers 6-244.2
6.111A.5 Examples 6-244.3
6.112 OA$TRM_INIT 6-245
6.112.1 Syntax 6-245
6.112.2 Description 6-245
6.112.3 Function Parameters 6-245
6.112.4 Function Qualifiers 6-245
6.113 OA$TRM_INQUIRE 6-246
6.113.1 Syntax 6-246
6.113.2 Description 6-246
6.113.3 Function Parameters 6-246
6.113.4 Function Qualifiers 6-246
6.114 OA$TRM_PORT_LINE 6-247
6.114.1 Syntax 6-247
6.114.2 Description 6-247
6.114.3 Function Parameters 6-247
6.114.4 Function Qualifiers 6-247
6.115 OA$TRM_PORT_LOG_ON 6-248
6.115.1 Syntax 6-248
6.115.2 Description 6-248
6.115.3 Function Parameters 6-248
6.115.4 Function Qualifiers 6-248
6.116 OA$TRM_PORT_OFF 6-249
6.116.1 Syntax 6-249
6.116.2 Description 6-249
6.116.3 Function Parameters 6-249
6.116.4 Function Qualifiers 6-249
6.117 OA$TRM_PORT_ON 6-250
6.117.1 Syntax 6-250
6.117.2 Description 6-250
6.117.3 Function Parameters 6-250
6.117.4 Function Qualifiers 6-250
6.118 OA$TRM_PORT_PRINT 6-251
6.118.1 Syntax 6-251
6.118.2 Description 6-251

6.118.3 Function Parameters 6-251
6.118.4 Function Qualifiers 6-251
6.119 OA$TXL_CLOSE 6-252
6.119.1 Syntax 6-252
6.119.2 Description 6-252
6.119.3 Function Parameters 6-252
6.119.4 Function Qualifiers 6-252
6.119.5 Special Features and Restrictions 6-252
6.120 OA$TXL_COMPILE 6-253
6.120.1 Syntax 6-253
6.120.2 Description 6-253
6.120.3 Function Parameters 6-253
6.120.4 Function Qualifiers 6-253
6.121 OA$TXL_DIRECTORY 6-254
6.121.1 Syntax 6-254
6.121.2 Description 6-254
6.121.3 Function Parameters 6-254
6.121.4 Function Qualifiers 6-254
6.122 OA$TXL_INSTALL 6-255
6.122.1 Syntax 6-255
6.122.2 Description 6-255
6.122.3 Function Parameters 6-255
6.122.4 Function Qualifiers 6-255
6.122.5 Special Features and Restrictions 6-255
6.123 OA$TXL_LIST 6-256
6.123.1 Syntax 6-256
6.123.2 Description 6-256
6.123.3 Function Parameters 6-256
6.123.4 Function Qualifiers 6-256
6.123.5 Examples 6-256
6.124 OA$TXL_OPEN 6-257
6.124.1 Syntax 6-257
6.124.2 Description 6-257
6.124.3 Function Parameters 6-257
6.124.4 Function Qualifiers 6-257
6.124.5 Special Features and Restrictions 6-258
6.125 OA$TXL_REMOVE 6-259
6.125.1 Syntax 6-259
6.125.2 Description 6-259
6.125.3 Function Parameters 6-259
6.125.4 Function Qualifiers 6-259
6.125.5 Special Features and Restrictions 6-259

6.126 OA$VAL_SET_VALID 6-260
6.126.1 Syntax 6-260
6.126.2 Description 6-260
6.126.3 Function Parameters 6-260
6.126.4 Function Qualifiers 6-260
6.126.5 Examples 6-261
6.127 PARSE_USER 6-262
6.127.1 Syntax 6-262
6.127.2 Description 6-262
6.127.3 Function Parameters 6-262
6.127.4 Function Qualifiers 6-263
6.127.5 Examples 6-263
6.128 PORT_ON, PORT_OFF 6-264
6.128.1 Syntax 6-264
6.128.2 Description 6-264
6.128.3 Function Parameters 6-264
6.128.4 Function Qualifiers 6-264
6.128.5 Examples 6-264
6.128.6 Special Features and Restrictions 6-264
6.129 PRINT_SCREEN 6-265
6.129.1 Syntax 6-265
6.129.2 Description 6-265
6.129.3 Function Parameters 6-266
6.129.4 Function Qualifiers 6-266
6.129.5 Examples 6-266
6.130 PROMPT 6-267
6.130.1 Syntax 6-267
6.130.2 Description 6-267
6.130.3 Function Parameters 6-268
6.130.4 Function Qualifiers 6-268
6.130.5 Examples 6-268
6.130.6 Special Features and Restrictions 6-269
6.131 PROMPT_ID 6-271
6.131.1 Syntax 6-271
6.131.2 Description 6-271
6.131.3 Function Parameters 6-272
6.131.4 Function Qualifiers 6-272
6.132 PURGE_FILE 6-273
6.132.1 Syntax 6-273
6.132.2 Description 6-273
6.132.3 Function Parameters 6-273
6.132.4 Function Qualifiers 6-274
6.132.5 Examples 6-274
6.132.6 Special Features and Restrictions 6-275

6.133 PUT_FIELD 6-277
6.133.1 Syntax 6-277
6.133.2 Description 6-277
6.133.3 Function Parameters 6-277
6.133.4 Function Qualifiers 6-278
6.133.5 Examples 6-278
6.133.6 Special Features and Restrictions 6-278
6.134 QUEUE_BATCH 6-279
6.134.1 Syntax 6-279
6.134.2 Description 6-279
6.134.3 Function Parameters 6-279
6.134.4 Function Qualifiers 6-279
6.134.5 Examples 6-288
6.134.6 Special Features and Restrictions 6-282
6.135 QUEUE_PRINT 6-284
6.135.1 Syntax 6-284
6.135.2 Description 6-284
6.135.3 Function Parameters 6-284
6.135.4 Function Qualifiers 6-284
6.135.5 Examples 6-287
6.135.6 Special Features and Restrictions 6-288
6.136 RENAME 6-289
6.136.1 Syntax 6-289
6.136.2 Description 6-289
6.136.3 Function Parameters 6-289
6.136.4 Function Qualifiers 6-289
6.136.5 Special Features and Restrictions 6-290
6.137 REPEAT 6-291
6.137.1 Syntax 6-291
6.137.2 Description 6-291
6.137.3 Function Parameters 6-291
6.137.4 Function Qualifiers 6-291
6.137.5 Examples 6-291
6.137.6 Special Features and Restrictions 6-292
6.138 SAVE 6-293
6.138.1 Syntax 6-293
6.138.2 Description 6-293
6.138.3 Function Parameters 6-293
6.138.4 Function Qualifiers 6-294
6.138.5 Examples 6-294
6.138.6 Special Features and Restrictions 6-294

6.139 SCRIPT 6-296
6.139.1 Syntax 6-296
6.139.2 Description 6-296
6.139.3 Function Parameters 6-296
6.139.4 Function Qualifiers 6-297
6.139.5 Examples 6-297
6.139.6 Special Features and Restrictions 6-297
6.140 SET_DEFAULT 6-298
6.140.1 Syntax 6-298
6.140.2 Description 6-298
6.140.3 Function Parameters 6-298
6.140.4 Function Qualifiers 6-298
6.140.5 Examples 6-299
6.140.6 Special Features and Restrictions 6-299
6.141 SET_MENU 6-300
6.141.1 Syntax 6-300
6.141.2 Description 6-300
6.141.3 Function Parameters 6-300
6.141.4 Function Qualifiers 6-300
6.141.5 Examples 6-301
6.141.6 Special Features and Restrictions 6-301
6.142 SHOW_FORM 6-302
6.142.1 Syntax 6-302
6.142.2 Description 6-302
6.142.3 Function Parameters 6-302
6.142.4 Function Qualifiers 6-302
6.143 SORT_LIST 6-303
6.143.1 Syntax 6-303
6.143.2 Description 6-303
6.143.3 Function Parameters 6-303
6.143.4 Function Qualifiers 6-305
6.143.5 Examples 6-305
6.143.6 Special Features and Restrictions 6-305
6.144 SPAWN 6-306
6.144.1 Syntax 6-306
6.144.2 Description 6-306
6.144.3 Function Parameters 6-306
6.144.4 Function Qualifiers 6-307
6.144.5 Special Features and Restrictions 6-307

6.145 STATISTICS 6-308
6.145.1 Syntax 6-308
6.145.2 Description 6-308
6.145.3 Function Parameters 6-308
6.145.4 Function Qualifiers 6-309
6.145.5 Examples 6-309
6.146 TEST_SPEC 6-310
6.146.1 Syntax 6-310
6.146.2 Description 6-310
6.146.3 Function Parameters 6-310
6.146.4 Function Qualifiers 6-310
6.146.5 Examples 6-311
6.146.6 Special Features and Restrictions 6-311
6.147 TIME 6-312
6.147.1 Syntax 6-312
6.147.2 Description 6-312
6.147.3 Function Parameters 6-312
6.147.4 Function Qualifiers 6-312
6.147.5 Examples 6-313
6.148 UNWIND 6-314
6.148.1 Syntax 6-314
6.148.2 Description 6-314
6.148.3 Function Parameters 6-314
6.148.4 Function Qualifiers 6-314
6.148.5 Examples 6-315
6.148.6 Special Features and Restrictions 6-315
6.149 VERSION 6-316
6.149.1 Syntax 6-316
6.149.2 Description 6-316
6.149.3 Function Parameters 6-316
6.149.4 Function Qualifiers 6-316
6.149.5 Examples 6-317
6.150 WAIT 6-318
6.150.1 Syntax 6-318
6.150.2 Description 6-318
6.150.3 Function Parameters 6-318
6.150.4 Function Qualifiers 6-319
6.150.5 Examples 6-319
6.150.6 Special Features and Restrictions 6-320

6.151 WPSPLUS$CBI_CLOSE_WINDOW 6-321
6.151.1 Syntax 6-321
6.151.2 Description 6-321
6.151.3 Function Parameters 6-321
6.151.4 Function Qualifiers 6-321
6.151.5 Examples 6-321
6.151.6 Special Features and Restrictions 6-321
6.152 WPSPLUS$CBI_CLOSE_WINDOW2 6-322
6.152.1 Syntax 6-322
6.152.2 Description 6-322
6.152.3 Function Parameters 6-322
6.152.4 Function Qualifiers 6-322
6.152.5 Examples 6-322
6.152.6 Special Features and Restrictions 6-322
6.153 WPSPLUS$CBI_OPEN_WINDOW 6-323
6.153.1 Syntax 6-323
6.153.2 Description 6-323
6.153.3 Function Parameters 6-323
6.153.4 Function Qualifiers 6-323
6.153.5 Examples 6-323
6.153.6 Special Features and Restrictions 6-324
6.154 WPSPLUS$CBI_OPEN_WINDOW2 6-325
6.154.1 Syntax 6-325
6.154.2 Description 6-325
6.154.3 Function Parameters 6-325
6.154.4 Function Qualifiers 6-325
6.154.5 Examples 6-326
6.154.6 Special Features and Restrictions 6-326
6.155 WPSPLUS_SPELL 6-327
6.155.1 Syntax 6-327
6.155.2 Description 6-327
6.155.3 Function Parameters 6-327
6.155.4 Function Qualifiers 6-327
6.155.5 Examples 6-327
6.155.6 Special Features and Restrictions 6-327
6.156 WRITE 6-328
6.156.1 Syntax 6-328
6.156.2 Description 6-328
6.156.3 Function Parameters 6-328
6.156.4 Function Qualifiers 6-329
6.156.5 Examples 6-329
6.156.6 Special Features and Restrictions 6-329

6.157 YESNO_PROMPT 6-330
6.157.1 Syntax 6-330
6.157.2 Description 6-330
6.157.3 Function Parameters 6-330
6.157.4 Function Qualifiers 6-331
6.157.5 Examples 6-331
6.157.6 Special Features and Restrictions 6-332
6.158 YESNO_PROMPT_ID 6-333
6.158.1 Syntax 6-333
6.158.2 Description 6-333
6.158.3 Function Parameters 6-333
6.158.4 Function Qualifiers 6-334
6.158.5 Special Features and Restrictions 6-334

Volume 3: Applications

Preface

7 The Subprocess

7.1 Introduction 7-1
7.2 Initializing the Subprocess 7-1
7.3 Communicating with the Subprocess Through Mailboxes 7-2
7.3.1 SYS$INPUT 7-4
7.3.2 OAMAILBOX and DCLMAILBOX 7-4
7.4 WRITE OAMAILBOX Syntax 7-6
7.5 Defining and Using Symbols 7-8
7.5.1 Creating DCL Symbols with /COPY and /PUT_SAVE 7-9
7.5.2 Creating DCL Symbols with Functions 7-10
7.6 Common DCL Symbols 7-12
7.6.1 Passing Command-Procedure Parameters to the Subprocess from ALL-IN-1 7-16
7.7 ALL-IN-1 Logicals 7-16
7.8 Running Command Procedures and Programs 7-19
7.8.1 Running Command Procedures 7-19
7.8.2 Obtaining User Input in a Program or Command Procedure 7-20

8 ALL-IN-1 Symbols

8.1 Introduction 8-1
8.2 Basic Symbol Types 8-2
8.2.1 Quoted String 8-2
8.2.2 Field Name 8-2
8.2.3 Permanent Symbol Table 8-2
8.2.4 Special Symbols 8-3
8.2.5 Local Symbol Table 8-4
8.2.6 Data Set References 8-4

8.3 Using Data Set References 8-5
8.3.1 Combining and Nesting DSRs 8-6
8.3.2 Specifying Alternate Keys with DSRs 8-6
8.3.3 Special Field Names 8-7
8.4 Symbol Substrings 8-10
8.5 Symbol Lists 8-11
8.6 Iterative Symbol Substitution 8-12
8.7 Specifying Date/Time Values 8-13
8.7.1 Creating and Updating Symbols and Data Sets 8-14
8.7.2 Getting Values from Symbols 8-15
8.7.3 Values Stored in Multiple Symbols 8-15
8.8 Special Symbols 8-15
8.9 Permanent Symbols 8-42

9 The File Cabinet

9.1 File Cabinet Concepts 9-2
9.2 File Cabinet Organization 9-4
9.2.1 OAUSER:DOCDB.DAT, the Personal File Cabinet Index 9-4
9.3 Document Attributes 9-8
9.3.1 The Document Attributes Files 9-9
9.3.2 Document Attributes 9-9
9.3.3 The System Pending File 9-12
9.4 Using the File Cabinet 9-13
9.4.1 Summary of CABINET Subfunctions 9-13
9.4.2 Accessing File Cabinet Data 9-15
9.4.3 The Shared Area 9-19
9.4.4 Shared Libraries 9-19
9.5 CABINET Subfunctions 9-20

10 Text Processing

10.1 Overview 10-1
10.2 Text Data Sets 10-2
10.3 Creating Files with the EDIT function 10-3
10.4 Features of the Default Editor 10-4
10.4.1 Editor Modes 10-4
10.4.2 Communicating with the Rest of ALL-IN-1 Through the Editor 10-10
10.4.3 Moving Within a Document 10-13
10.4.4 Moving Text Within a Document 10-14
10.4.5 Including Outside Text into a Document 10-16
10.4.6 Creating New Files While in the Editor 10-17
10.4.7 Customizing the Default Editor 10-18

10.5 Features of the WPS-PLUS Editor 10-21
10.6 List Processing 10-21
10.7 Formatting and Outputting Text 10-28

11 Electronic Messaging

11.1 Introduction 11-1
11.2 Electronic Messaging System Concepts 11-1
11.2.1 Current Message 11-1
11.2.2 Message Header and Text 11-2
11.2.3 Mail Interfaces 11-2
11.2.4 Mail Folders 11-2
11.2.5 Nicknames and Distribution Lists 11-3
11.2.6 Mail Delivery Mechanism 11-3
11.2.7 Alternative Message Transport Systems (MTS) 11-5
11.3 Special Symbols 11-9
11.4 Customizing the Message Header and Show-Message Displays 11-9
11.5 MAIL Subfunctions 11-10
11.5.1 ANSWER 11-11
11.5.2 ATTACH 11-13
11.5.3 AUTO_FORWARD 11-13
11.5.4 AUTO_REPLY 11-14
11.5.5 CANCEL 11-14
11.5.6 CANCEL_AUTO_REPLY 11-15
11.5.7 CC 11-15
11.5.8 CLOSE_MESSAGE 11-16
11.5.9 CREATE 11-16.1
11.5.10 DEFAULT_MESSAGE 11-19
11.5.11 DEFER 11-20
11.5.12 DELETE 11-20
11.5.13 DELIVERY_RECEIPT 11-21
11.5.14 DETACH 11-22
11.5.15 DIST_COPY 11-22
11.5.16 DIST_CREATE 11-23
11.5.17 DIST_DELETE 11-23
11.5.18 DIST_EDIT 11-24
11.5.19 DIST_LIST 11-24
11.5.20 EDIT 11-25
11.5.21 EXCHANGE_CURMES 11-26
11.5.22 EXPAND_DIST_LIST 11-26
11.5.23 FILE_ATTACHMENT 11-27

11.5.24 FILE_BODY 11-28
11.5.25 FILE_MESSAGE 11-29
11.5.26 FORWARD 11-30
11.5.27 FORWARDABLE 11-32
11.5.28 FUNCTION 11-32
11.5.29 GET 11-33
11.5.30 INDEX 11-33
11.5.31 MAIL_DOCUMENT 11-34
11.5.32 NEXT 11-36
11.5.33 NO_DELIVERY_RECEIPT 11-37
11.5.34 NON_FORWARDABLE 11-37
11.5.35 ORIGINAL 11-37
11.5.36 OUTBOX 11-38
11.5.37 POP_CURMES 11-38
11.5.38 PRINT 11-39
11.5.39 PRIORITY 11-40
11.5.40 PUSH_CURMES 11-41
11.5.41 READ 11-42
11.5.42 READ_RECEIPT 11-43
11.5.43 SEND 11-44
11.5.44 SHOW 11-46
11.5.45 STATUS 11-46
11.5.46 SUBJECT 11-47
11.5.47 TEXT 11-48
11.5.48 TO 11-48
11.5.49 VAXMAIL_CHECK 11-49
11.6 Electronic Messaging Data Set References 11-50

12 External Communications

12.1 Overview 12-1
12.2 Multinode ALL-IN-1 Applications 12-1
12.3 The Communications Control Subsystem 12-2
12.3.1 Communications Control Concepts 12-2
12.3.2 CXP Symbols 12-2
12.3.3 CXP Subfunctions 12-5
12.3.4 CXP Subfunctions that Differ from Control Document Commands 12-9
12.3.5 Using Communication Control Commands 12-11
12.3.6 Example of Customizing Communications Control 12-12

13 Time Management

13.1 Introduction 13-1
13.2 Organization 13-1
13.2.1 Time Management Activities 13-2
13.2.2 Subfunctions and the Current CALENDAR Date/Time 13-2
13.2.3 CALENDAR Files 13-3
13.3 CALENDAR Subfunctions 13-3
13.3.1 CALENDAR ADD_ATTENDEE 13-3
13.3.2 CALENDAR CANCEL APPOINTMENT 13-4
13.3.3 CALENDAR CANCEL MEETING 13-4
13.3.4 CALENDAR CLEAR ATTENDEES 13-5
13.3.5 CALENDAR CONVERT APPOINTMENTS 13-5
13.3.6 CALENDAR CONVERT ACTION_ITEMS 13-5
13.3.7 CALENDAR CONVERT MEETINGS 13-6
13.3.8 CALENDAR DISPLAY ANSWER 13-6
13.3.9 CALENDAR DISPLAY APPOINTMENT 13-7
13.3.10 CALENDAR DISPLAY GRAPH 13-8
13.3.11 CALENDAR DISPLAY MEETING 13-9
13.3.12 CALENDAR DISPLAY MONTH 13-9
13.3.13 CALENDAR DISPLAY NEXT APPOINTMENT 13-10
13.3.14 CALENDAR DISPLAY NEXT MEETING 13-11
13.3.15 CALENDAR DISPLAY NEXT REPLY 13-11
13.3.16 CALENDAR DISPLAY REPLY 13-12
13.3.17 CALENDAR INITIALIZE FILES 13-13
13.3.18 CALENDAR INITIALIZE MONTH 13-13
13.3.19 CALENDAR MAIL_TO_ATTENDEES 13-14
13.3.20 CALENDAR MAKE_ATTENDEE_SYMBOLS 13-14
13.3.21 CALENDAR MAKE_MEETING_SYMBOLS 13-15
13.3.22 CALENDAR PRINT ALL 13-16
13.3.23 CALENDAR PRINT CONFLICTS 13-16
13.3.24 CALENDAR PRINT CUSTOM 13-17
13.3.24A CALENDAR PRINT DAY_REMIND 13-17
13.3.25 CALENDAR PRINT DAY_SCHEDULE 13-18
13.3.26 CALENDAR PRINT MONTH 13-18
13.3.26A CALENDAR PRINT TODO 13-19
13.3.27 CALENDAR PRINT TWOCAL 13-19
13.3.28 CALENDAR PRINT WEEK_REMINDERS 13-20
13.3.29 CALENDAR PRINT WEEK_SCHEDULE 13-20
13.3.30 CALENDAR PURGE 13-20.1

13.3.31 CALENDAR REMOVE_ATTENDEES 13-20.1
13.3.32 CALENDAR REPLY_TO_MEETING 13-21
13.3.33 CALENDAR RESCHEDULE APPOINTMENT 13-21
13.3.34 CALENDAR RESCHEDULE MEETING 13-22
13.3.35 CALENDAR SCAN 13-22
13.3.36 CALENDAR SCHEDULE APPOINTMENT 13-23
13.3.37 CALENDAR SCHEDULE MEETING 13-23
13.3.38 CALENDAR SET ALTERNATE_USER 13-24
13.3.39 CALENDAR SET DATE 13-25
13.3.40 CALENDAR SET NETWORK OFF 13-26
13.3.41 CALENDAR SET NETWORK ON 13-26
13.3.42 CALENDAR VERSION 13-27
13.4 Calendar File Layouts 13-27
13.4.1 User File Layout 13-27
13.4.2 Meeting File Layout 13-29
13.4.3 Attendee File Layout 13-30
13.5 CALENDAR Data Sets 13-30

14 The Help and Message Facilities

14.1 Introduction 14-1
14.2 Help Facility 14-1
14.2.1 The HELP Function 14-1
14.2.2 The Help Library 14-6
14.2.3 Maintaining, Updating, and Switching Between Help Libraries 14-18
14.2.4 Help at a Prompt 14-22
14.3 Message Facility 14-22
14.3.1 The Message File 14-23
14.3.2 Updating the Message File 14-23
14.3.3 Message Severity and the Message Buffer 14-24
14.3.4 Help on Error Messages 14-26

15 The Script Facility

15.1 Overview 15-1
15.2 Scripts 15-1
15.2.1 Script Modes 15-2
15.2.2 Interactive Directives and the Pseudo-Input Buffer 15-3
15.2.3 Exiting from Scripts 15-4
15.2.4 ALL-IN-1 TXLs 15-5
15.2.5 Script-File Format 15-7

15.3 Script Directives 15-9
15.4 Script Directives as Functions 15-31
15.5 Sample Script 15-32
15.6 User Defined Processes 15-33
15.7 Running ALL-IN-1 Indirectly or in Batch Mode 15-33
15.8 Special Features and Restrictions 15-33
15.8.1 Creating a Script with a Script 15-33
15.8.2 UDPs for the Desk Calculator 15-34

16 Application Development Tools and Design Guidelines

16.1 Introduction 16-1
16.1.1 Where to Start 16-2
16.2 The Forms Development Subsystem 16-2
16.2.1 Using Form Libraries 16-9
16.2.2 Testing Newly Created Forms 16-11
16.3 The Programming Functions Subsystem 16-12
16.4 Other Development Tools 16-14
16.4.1 Debugging Features 16-15
16.4.2 Using the Subprocess Interactively 16-15
16.4.3 Using the Editor 16-16
16.4.4 Using the GET Function 16-17
16.4.5 Using View Forms 16-18
16.5 Application Design Guidelines 16-18
16.5.1 Standard Applications 16-19
16.5.2 Standalone Applications 16-20
16.5.3 User-Defined Functions 16-21
16.6 Performance Considerations 16-22
16.7 Documenting a Customized System 16-23
16.7.1 User Documentation 16-23
16.7.2 Help Documentation 16-24

A Exercises

A.1 Introduction A-1
A.2 Exercise 1 — Using Standard ALL-IN-1 Subsystems A-2
A.2.1 Objective A-2
A.2.2 Preparation A-2
A.2.3 Exercise A-2
A.2.4 Summary A-3
A.2.5 References A-3

A.3 Exercise 2 — Modifying the User Profile A-3
A.3.1 Objective A-3
A.3.2 Preparation A-3
A.3.3 Exercise A-3
A.3.4 Summary A-4
A.3.5 References A-4
A.4 Exercise 3 — Advanced User Tasks A-4
A.4.1 Objective A-4
A.4.2 Preparation A-4
A.4.3 Exercise A-4
A.4.4 Summary A-5
A.4.5 References A-5
A.5 Exercise 4 — Modifying the User Profile (Programmer Level) A-5
A.5.1 Objective A-5
A.5.2 Preparation A-6
A.5.3 Exercise A-6
A.5.4 Summary A-6
A.5.5 References A-7
A.6 Exercise 5 — Accessing the Subprocess A-7
A.6.1 Objective A-7
A.6.2 Preparation A-7
A.6.3 Exercise A-7
A.6.4 Summary A-8
A.6.5 References A-8
A.7 Exercise 6 — Using Functions Interactively A-8
A.7.1 Objective A-8
A.7.2 Preparation A-8
A.7.3 Exercise A-8
A.7.4 Summary A-9
A.7.5 References A-9
A.8 Exercise 7 — Using the Hard Copy Interface A-9
A.8.1 Objective A-9
A.8.2 Preparation A-9
A.8.3 Exercise A-10
A.8.4 Summary A-11
A.8.5 References A-11
A.9 Exercise 8 — Using Development Tools A-11
A.9.1 Objective A-11
A.9.2 Preparation A-11
A.9.3 Exercise A-11
A.9.4 Summary A-12
A.9.5 References A-12

A.10 Exercise 9 — Modifying a Menu A-12
A.10.1 Objective A-12
A.10.2 Preparation A-12
A.10.3 Exercises A-12
A.10.4 Summary A-13
A.10.5 References A-13
A.11 Exercise 10 — Creating an Entry Form for an Indexed File A-13
A.11.1 Objective A-13
A.11.2 Preparation A-14
A.11.3 Exercise A-14
A.11.4 Summary A-15
A.11.5 References A-15
A.12 Exercise 11 — Creating a Select Form A-15
A.12.1 Objective A-15
A.12.2 Preparation A-15
A.12.3 Exercise A-15
A.12.4 Summary A-16
A.12.5 References A-16
A.13 Exercise 12 — Modifying a Script A-16
A.13.1 Objective A-16
A.13.2 Preparation A-17
A.13.3 Exercise A-17
A.13.4 Summary A-17
A.13.5 References A-17
A.13.6 Answers A-18
A.14 Exercise 13 — Creating a Shared Template A-19
A.14.1 Objective A-19
A.14.2 Preparation A-19
A.14.3 Exercise A-19
A.14.4 Summary A-20
A.14.5 References A-20
A.15 Exercise 14 — Using DATATRIEVE A-20
A.15.1 Objective A-20
A.15.2 Preparation A-21
A.15.3 Exercise A-21
A.15.4 Summary A-22
A.15.5 References A-22

B ALL-IN-1 Hard Copy

B.1 ALL-IN-1 Hard Copy B-1
B.1.1 Intended Use B-1
B.1.2 Terminal Support B-2

B.1.3 Function Keys B-3
B.1.4 Appearance on the Screen B-3
B.1.5 /HARD Qualifiers B-4
B.1.6 Function-Key Entry B-5
B.1.7 Field-Full Condition B-5

C Data Sets

C.1 Overview C-1
C.2 Special Data Sets C-2
C.2.1 OA$DATE C-2
C.2.2 OA$DIR C-3
C.2.3 OA$DIR_PARSE C-6
C.2.4 OA$EDITOR C-7
C.2.5 OA$MAIDES C-7
C.2.6 OA$MAIL_ADDRESS C-8
C.2.7 OA$MAILPRIO C-8
C.2.8 OA$TABLE C-8
C.2.9 OA$YESNO C-9
C.2.10 OA$YN C-9
C.3 Calendar Validation Data Sets C-9
C.3.1 CAL$ATTENDEE C-9
C.3.2 CAL$CALENDAR C-10
C.3.3 CAL$END_TIME_CHECK:key C-10
C.3.4 CAL$MEETING C-10
C.3.5 CAL$MTG_TIME_CHECK C-10
C.3.6 CAL$SET_DATE C-10
C.3.7 CAL$SET_OWNER C-11
C.3.8 CAL$SET_SECOND C-11
C.3.9 CAL$TIME_CHECK C-11
C.4 Calendar Scrolling Data Sets C-11
C.4.1 CAL$SCROLL_BOTH C-12
C.4.2 CAL$SCROLL_CONFLICTS C-12
C.4.3 CAL$SCROLL_CUSTOM C-12
C.4.4 CAL$SCROLL_DAY C-12
C.4.5 CAL$SCROLL_TWOCAL C-12
C.4.6 CAL$SCROLL_WEEK_REM C-13
C.4.7 CAL$SCROLL_WEEK_SCH C-13
C.5 General Scrolling Data Sets C-13
C.5.1 SCROLL$FUNCTION C-13
C.6 Data Set Statistics C-13

D Mnemonics for Script Input

Glossary

Figures

How To Use this APR xiv
ALL-IN-1 Bookshelf xvi
1-1 The VAX Information Architecture 1-2
1-2 ALL-IN-1 and the VAX Information Architecture 1-4
1-3 Form and Function 1-5
2-1 Primary ALL-IN-1 Flow-Control Framework 2-4
2-2 Flow Control Path: Display Main Menu 2-6
3-1 Menu Processing Flow Control 3-30
3-2 Form WPSEARCH and assumed Named Data 3-62
5-1 A Subsystem Menu 5-2
5-2 A Select Form 5-4
5-3 A Create Form 5-5
5-4 A Secondary Menu Form 5-8
7-1 Flow Control Between the Subprocess and Main Process 7-3
8-1 Obtaining Data with a DSR 8-9
9-1 File Cabinet Organization 9-13
10-1 Key Numbers for the Default Editor 10-12
10-2 Entry Form FORMAT (Document Handling and File Processing) 10-29
16-1 Forms Development Menu 16-3
16-2 Programming Functions Menu 16-13
16-3 Level-1 HELP Module Template 16-28
16-4 Level-2 HELP Module Template 16-29
16-5 Level-3 HELP Module Template 16-29

Tables

3-1a Form AIE: Directives 3-40
3-1b Form AIE: Field Information 3-41
3-2 File OA$LIB:ACTITEM.FDL 3-43
4-1 SCROLL$FUNCTION Actions 4-37
4-2 Control-Function Calls to Internal Terminator Functions 4-67
4-3 Terminator Functions, Dispositions, and OA$FORM_DISPOSE 4-68
4-4 OA$FORM_TERMINATOR Values 4-69

6-1 COMPUTE Operators and Operands 6-38
6-2 DATE_CONVERT Parameters 6-49
6-3 Relational Operators 6-85
6-4 Boolean Expressions 6-86.1
6-5 Recognized Editor Styles 6-131
7-1 ALL-IN-1 Subprocess Logicals 7-18
8-1 Special Symbols 8-16
8-2 Permanent Symbols 8-42
9-1a Document Attributes 9-10
9-1b Extended Attributes 9-11
10-1 Recognized Editor Styles 10-3
10-2 File Processors 10-32
B-1 Supported Terminals B-2
C-1 Data Set Statistics (OA$DSA_CACHE_STATUS output) C-14

Preface

Purpose of this Guide

The *ALL-IN-1 Office Menu Application Programmer's Reference* (APR) gives a detailed overview of ALL-IN-1 and presents the tools available to the applications programmer for modifying or adding new applications. The APR also provides Application Design Guidelines and Exercises.

Intended Audience

The APR is written for the applications programmer familiar with VAX/VMS and its layered products. References to other documents that provide additional information are presented in the ALL-IN-1 Bookshelf illustrated in this Preface. References are also noted, where relevant, throughout this APR.

Before creating an application, you should be able to:

- Understand the following VAX/VMS concepts:
  - DCL
  - directory
  - file type
  - process
  - protection
  - shareable image

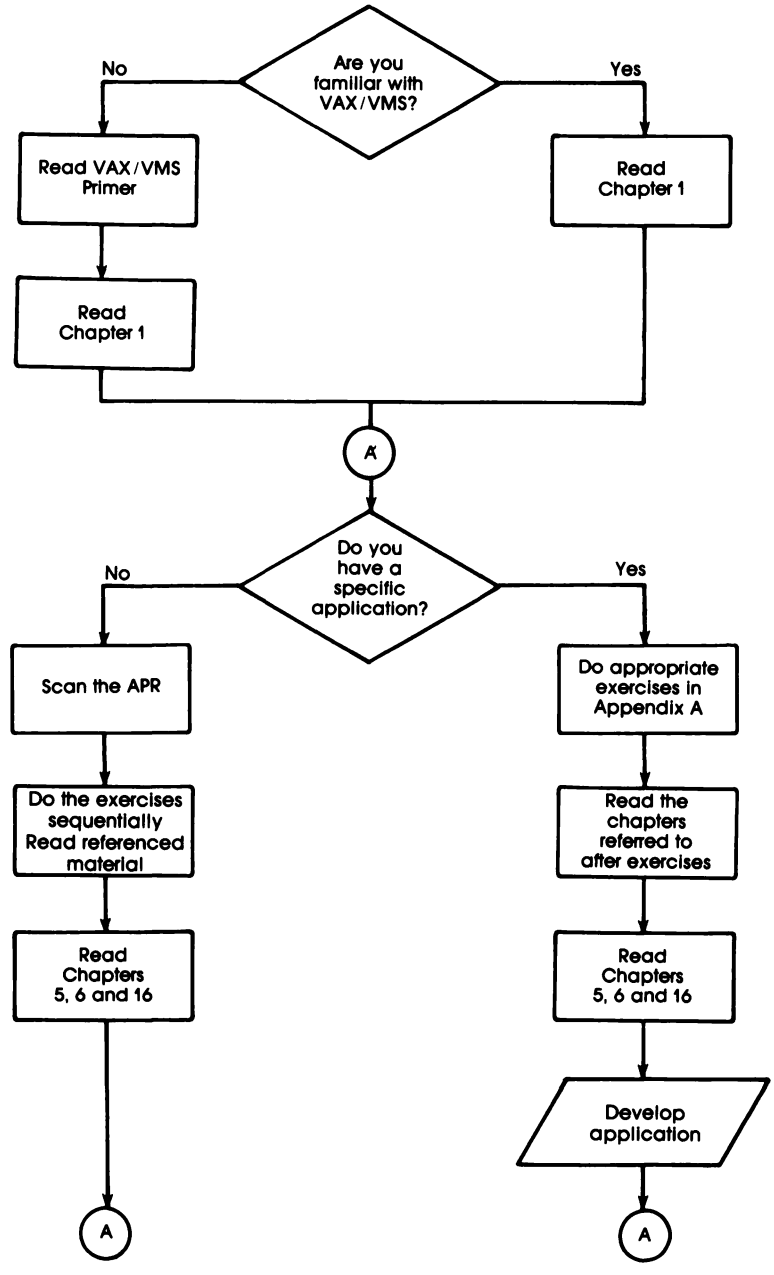
  - library — subprocess
  - logical — symbol
  - privilege — user identification code

- Login to a VMS system.
- Edit a file using the EDT text editor.
- Create a simple DIGITAL Command Language (DCL) command procedure.
- Create a Forms Management System (FMS) form, and understand the meaning and use of Named Data within a form.
- Create and access subdirectories.
- Create DCL symbols and assign logical names.

If you are unfamiliar with these concepts, refer to the flowchart on the following page and to the Glossary at the end of this guide. The flowchart may also be useful if you are familiar with VAX/VMS concepts.

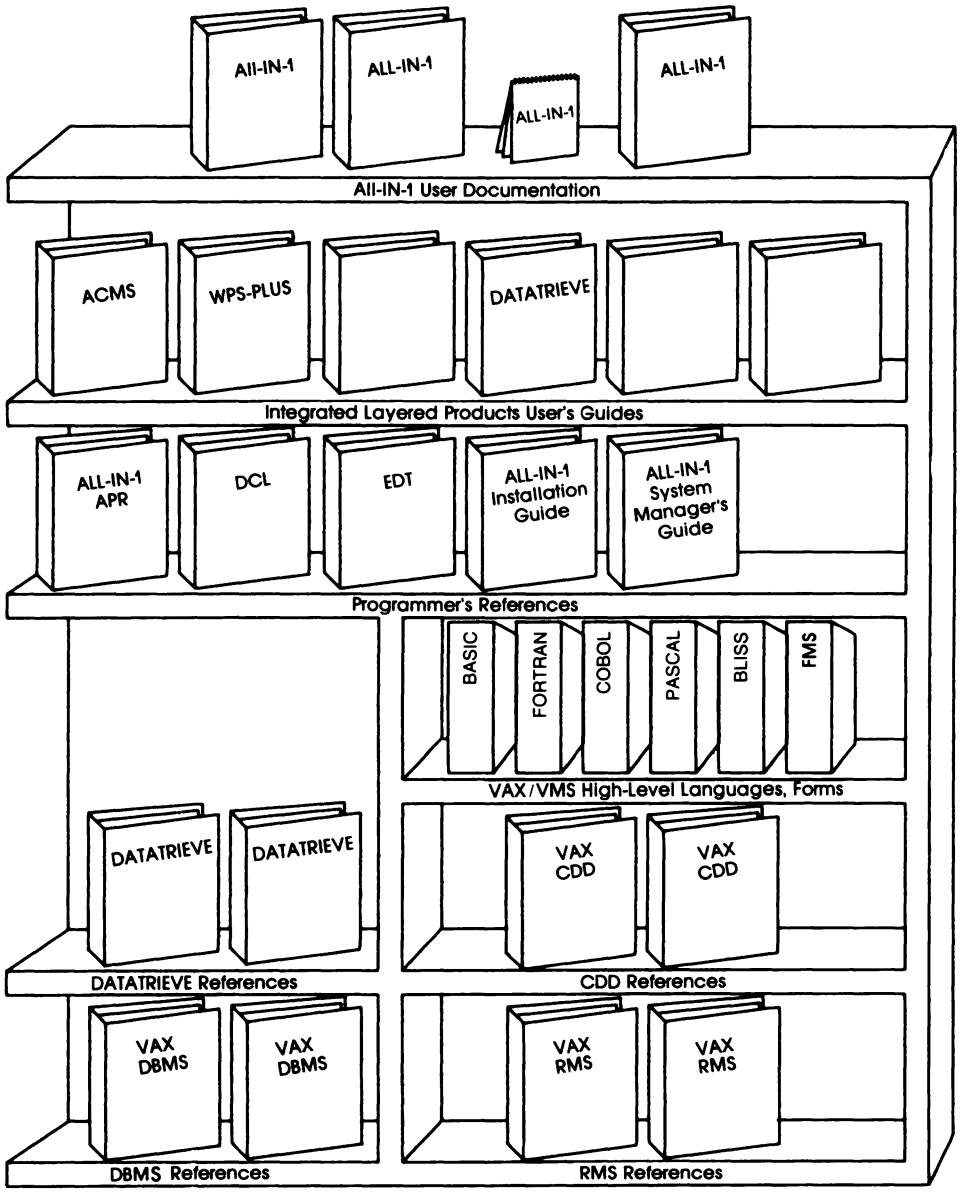
How the APR is Organized

The APR has three volumes: Volume 1, Flow Control, covers the kernel of ALL-IN-1. Volume 2, ALL-IN-1 Functions, describes the functions. These are calls to ALL-IN-1 facilities to request that a specific action be performed. Volume 3, Applications, covers those closely integrated applications that use the Flow Control facilities.



MK-02000-00

How to Use this APR



ALL-IN-1 Bookshelf

This Bookshelf contains all the documents referenced throughout the APR. These documents are also listed in the Related Documents section that follows.

Related Documents

The following documents are part of the ALL-IN-1 Office Menu Documentation Set:

User Documentation

ALL-IN-1 Office Menu Getting Started Guide
ALL-IN-1 Office Menu User's Reference
Volume 1
Volume 2
ALL-IN-1 Office Menu Quick Lookup

Programmer Documentation

ALL-IN-1 Office Menu Installation Guide
ALL-IN-1 Office Menu System Manager's Guide
ALL-IN-1 Office Menu Release Notes
ALL-IN-1 Office Menu Read Me First
ALL-IN-1 Office Menu Application Programmer's Reference
Volume 1
Volume 2
Volume 3
ALL-IN-1 Office Menu Programmer's Mini-Reference

Customization Documentation

ALL-IN-1 Documenter's Kit

ALL-IN-1 Office Menu Style Guide
ALL-IN-1 Office Menu Writer's Guide
Tape of Getting Started Guide and User's Reference

The following documents cover VAX/VMS concepts and VMS layered products that are integrated into or accessible from ALL-IN-1.

Application Control and Management System (ACMS)

VAX ACMS Application Management Guide

DATATRIEVE (DTR)/Database (DBMS)/Common Data Dictionary (CDD)

VAX CDD Utilities Reference Manual
Introduction to VAX DATATRIEVE
VAX DATATRIEVE User's Guide
VAX DATATRIEVE Reference Manual
VAX DATATRIEVE Guide to Using Graphics
VAX DBMS Introduction to Data Manipulation
VAX DBMS Database Administration Reference Manual
VAX DBMS Programming Reference Manual

WPS-PLUS Editor

WPS-PLUS/VMS G g Started
WPS-PLUS/VMS Editor Functions
WPS-PLUS/VMS List and Sort Processing
WPS-PLUS/VMS Quick Lookup

EDT Editor

VAX EDT Reference Manual

Forms Management System (FMS)

VAX-11 FMS Utilities Manual

Programming Languages

VAX-11 BASIC Language Reference Manual
VAX-11 BLISS-32 Language Reference Manual
VAX-11 COBOL Language Reference Manual
VAX-11 FORTRAN Language Reference Manual
VAX-11 PASCAL Language Reference Manual
VAX-11 PL\1 Language Reference Manual

VMS Operating System

VAX/VMS Document Set
VAX/VMS Release Notes

Conventions

The following conventions are used throughout this document:

Dot Matrix	Indicates prompts and messages from the computer.
lowercase	Lowercase text in an ALL-IN-1 function call, or in a DCL command or command procedure, indicates that you substitute a valid expression in that place.
UPPERCASE	Indicates keyboard keys and Gold functions.
Enter	Indicates information the user puts into a menu or form.
Press	Indicates that the user invokes a single keyboard command (for example, press EXIT SCREEN).
Type	Indicates characters the user puts into a document, program module, or mail message from the keyboard.
Gold X	Indicates that the user presses the Gold key and then the X key to invoke a keyboard command.
CTRL/X	Indicates that the user holds down the CTRL key while pressing X to invoke a keyboard command.

[ ]	Indicates an optional expression in ALL-IN-1 function calls and DCL command strings.
. . .	Indicates that the example contains more data than is shown.
. . .	Indicates that the user can enter additional parameter values or information.
< >	Are used in ALL-IN-1 boiler plate forms to set off areas that contain data to be processed (for example, <OA$TIME>).
|	Indicates alternatives (for example, /INT_CHARACTER={A|B|D} means A, or B, or D).

**Pages missing
from original**

2 Find the nearest alphabetical command in the table of keywords in module OA$BUILD:OASDF.BLI which is used to dispatch to ALL-IN-1 functions. Insert the function name and the name of the top-level routine that performs the function. The syntax for the keyword table is:

```
<'function',routine>,
```

where function is the function name and routine is the name of the routine that ALL-IN-1 calls when you call the function. The commas, single quotes, and angle brackets are required syntax.

This step adds the function name to the keyword table, creates an external routine reference to the routine specified, and associates the keyword with the external reference.

3 Compile the OASDF.BLI module and add it back into the object module library:

```
$ BLISS OA$BUILD:OASDF
$ LIBRARY OA$BUILD:OALIBR OASDF
```

4 Create the module, in your choice of language, that contains the routine specified in the keyword table. ALL-IN-1 calls it with one parameter: a string descriptor containing the rest of the command line following the function name. For example, if the function name is CONVERT and the command line is

```
CONVERT X to Y
```

then the input parameter to the routine is the string X to Y. Your routine then parses this line to determine the specific action to take, performs its function, closes out any files, and then returns.

5 Compile your module.

6 Modify the OA$BUILD:A1LINK.COM procedure that links ALL-IN-1 to include a reference to the new module.

7 Link ALL-IN-1 by executing OA$BUILD:A1LNKDRV.COM.

NOTE: Because ALL-IN-1 uses backslashes to separate functions in function lists, do not include them in the syntax of your subfunction calls. Also, if you include single quotes (apostrophes) in the syntax, the function will not work properly when called from the subprocess, where DCL interprets single quotes as symbol substitution characters.

6.6 Symbol Substitution in Functions

ALL-IN-1 symbols take several forms, ranging from quoted strings to data set references. Some functions — in particular, the GET function — recognize all types of symbols. Other functions, such as LIST, recognize only literals.

In general, whenever a function takes a symbol as an argument, the exact type or types of symbol(s) are described. Also, any function that does not recognize the full range of ALL-IN-1 symbols can be called with GET OA$FUNCTION or another special GET output destination that builds a function call from symbol lists.

The remainder of this chapter is an alphabetical listing of the ALL-IN-1 control functions and those internal functions that can be called like control functions. Like DCL commands, functions may be abbreviated to their shortest nonambiguous version. However, this practice is discouraged for other than interactive calls because new functions added in future releases may cause these abbreviations to become ambiguous.

6.7 ACMSTASK

6.7.1 Syntax

ACMSTASK appl_name task_name [,sel_string]

6.7.2 Description

The ACMSTASK function provides an interface to the current version of the Application Control and Management System (ACMS). When ACMS is available under ALL-IN-1, ACMSTASK communicates with this layered product, passing it an application name, a task name and a selection string that you provide as the ACMSTASK parameters.

You can call ACMSTASK from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

When you call ACMSTASK from ALL-IN-1, users bypass the ACMS menu interface and execute tasks directly from ALL-IN-1.

6.7.3 Function Parameters

appl_name

This required ACMSTASK parameter is the name of an ACMS application in which the task, specified by the next parameter, resides.

task_name

The single, required ACMSTASK parameter is the name of an ACMS task to execute. The VAX ACMS documentation set will show you how to define ACMS applications and task.

sel_string

This optional parameter includes all characters after the task_name except for the leading spaces. You can refer to the VAX ACMS documentation set for the proper use of this string.

6.7.4 Function Qualifiers

None.

6.7.5 Examples

The following sequence describes how to create a menu that accesses ACMS:

1. Create a menu form using the Forms Development Create option. During the create, specify the standard MENU form to use as a template. Specify an appropriate forms library where the form is to be stored.
2. Develop the screen using the FMS Layout option. Create ACMS-related menu options. For example:

```
                    ACMS Menu

A Option A

B Option B

C Option C

Enter selection and press RETURN
```

3 Create the associated Named Data with the FMS DATA option. The ACMSTASK function is called from Named Data. For example:

Named Data

```
1 Name .TYPE
MENU /CHOICE = CHOICE/CLEAR

2 Name A
ACMSTASK appl_1 task_a THE SELECTION STRING

3 Name B
ACMSTASK appl_1 task_b "The selection string includes the
quotes"

4 Name C
ACMSTASK appl_2 task_c
```

4 Save the form in the specified library.

Test the form. Enter the form name in the choice field of an ALL-IN-1 menu. Select the ACMS menu options.

ACMSTASK can also be called from command procedures running in the subprocess through OAMAILBOX.

6.7.6 Special Features and Restrictions

The ALL-IN-1/ACMS interface must be linked into the ALL-IN-1 image before ACMSTASK can be called. The Version 2.0 link command procedure, as described in the *Installation Guide*, contains provisions for integrating ACMS and ALL-IN-1.

6.8 ADE

6.8.1 Syntax

ADE [ADE command]

6.8.2 Description

The ADE function provides an interface to the current version of the VMS ADE product (ADE). When ADE is available under ALL-IN-1, the ADE function communicates with this layered product, passing it the command string that you provide as the parameter.

You can call ADE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.8.3 Function Parameters

command

The single, optional parameter is any VMS ADE command string.

6.9 AI_CHECK

6.9.1 Syntax

AI_CHECK

6.9.2 Description

The AI_CHECK function checks OAUSER:ACTITEM.DAT, the action items file, for open action items and displays the number of open items on line 24. An open action item is any record in ACTITEM.DAT:

1. For which AI_TYPE equals "A" (for Action item)
2. For which AI_STATUS equals "O" (for Open)
3. For which a scheduled date falls on or before the current system date

You can call AI_CHECK from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.9.3 Function Parameters

None.

6.9.4 Function Qualifiers

None.

6.9.5 Examples

Interactively:

```
<AI_CHECK
```

The number of open action items is displayed on line 24 and the user can continue immediately (without pressing RETURN). The line containing the number of open action items disappears from the screen when any key is pressed.

6.9.6 Special Features and Restrictions

ALL-IN-1 calls AI_CHECK automatically during initialization if the User Profile ACTITEM field contains a Y.

6.10 APPEND

6.10.1 Syntax

APPEND source__file__symbol destination__file__symbol

6.10.2 Description

The APPEND function attaches text files together. The syntax of this function is similar to the DCL APPEND command.

You can call APPEND from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.10.3 Function Parameters

source__file__symbol

The source__file__symbol is any ALL-IN-1 symbol which contains a VMS file name. This file is appended to the output file.

destination__file__symbol

The destination__file__symbol is any ALL-IN-1 symbol which contains a VMS file name. The input file is appended to this file.

6.10.4 Examples

In Named Data:

Named Data

```
1 Name AP
APPEND INFILE #OUTFILE__\GET OA$FUNCTION="DISPLAY Your File has

2 Name AP
 been appended to" #OUTFILE
```

In this example, when a user enters a file into the field INFILE, it is appended to the end of the file which is in the symbol #OUTFILE.

6.11 ATTACH

6.11.1 Syntax

ATTACH [/IDENT = proc-id]

6.11.2 Description

ATTACH transfers control to a detached subprocess, which has been spawned (see the SPAWN function) during the session. The function can also transfer control from one subprocess to another and back to the main process.

You call ATTACH from the following locations:

- In Named Data
- Interactively
- In scripts

6.11.3 Function Parameters

None.

6.11.4 Function Qualifiers

/IDENT = proc-id

The /IDENT qualifier specifies the process identification of the process you wish to attach to.

6.11.5 Example

<ATTACH/IDENT = 30019

This command switches control of the terminal from the current process to a process having the Process Identification 30019.

6.11.6 Special Features and Restrictions

You must use attach again to return to the main process. You cannot use EXIT to leave a subprocess.

See the VAX/VMS documentation set for further information on ATTACH and Subprocesses.

6.12 BYE

6.12.1 Syntax

BYE

6.12.2 Description

The BYE function terminates the ALL-IN-1 image, closes all open files, and then logs out the VMS process. Use the BYE function to exit from ALL-IN-1 and VMS in one step.

You can call BYE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.12.3 Function Parameters

None.

6.12.4 Function Qualifiers

None.

6.12.5 Examples

Interactively:

```
<BYE
```

From Named Data:

Named Data

```
1 Name .TYPE
MENU/CHOICE = CHOICE/CLEAR

2 Name LO
BYE

3 Name EX
EXIT
```

This call from Named Data illustrates use of the BYE function to exit VMS from an ALL-IN-1 menu.

6.12.6 Special Features and Restrictions

A statement in OAINI.COM assigns the BYE function call to the DCL synonym BYE. OAINI.COM is the command procedure, either in OA$LIB or OAUSER, that runs when ALL-IN-1 creates the subprocess. Thus, entering BYE at interactive DCL level in the subprocess logs you out of VMS.

However, calling the BYE function deletes the process, regardless from where it is called. Therefore, if you run ALL-IN-1 from a command procedure, calling BYE aborts that command procedure.

6.13 CABINET

6.13.1 Syntax

CABINET subfunction

6.13.2 Description

The CABINET function calls the File Cabinet facility, a file-management system used by applications such as Word and Document Processing and Electronic Messaging. CABINET takes subfunctions as its parameters. These subfunctions create, delete, share, and store information about documents, mail messages, or any other type of file.

You can call CABINET from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.13.3 Subfunction Descriptions

The CABINET subfunctions are covered in Chapter 9.

6.14 CALENDAR

6.14.1 Syntax

CALENDAR subfunction

6.14.2 Description

The CALENDAR function provides a large part of the functionality for the Time Management application. The CALENDAR subfunctions allow a user to schedule, view, modify, display, output, and purge appointments, meetings, and other time-related activities.

You can call CALENDAR from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.14.3 Subfunction Descriptions

Chapter 13 covers the Time Management application and the CALENDAR subfunctions in detail.

6.14A CHECK__DISKQUOTA

6.14A.1 Syntax

CHECK__DISKQUOTA [/qual] [vms__user][,dev][,threshold][,quota][,disk]

6.14A.2 Description

The CHECK__DISKQUOTA function tests that a user's unused disk quota exceeds a certain amount, or that a certain amount of space remains on a specified disk. If the quota is below the given threshold, CHECK__DISKQUOTA either outputs a warning message, or updates a named symbol. You can call CHECK__DISKQUOTA from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.14A.3 Function Parameters

vms__user

This is the VMS user name whose account quotas are checked. If this parameter is omitted, then the current VMS user name is used.

dev

This is the name of the device on which quotas are checked. If this parameter is omitted, then the current device is used.

threshold

This is the number of blocks that CHECK__DISKQUOTA checks against. If this parameter is omitted, CHECK__DISKQUOTA checks that the user has at least 100 blocks remaining.

quota

This is a symbol used to return the amount of unused quota remaining to the user. If this parameter is omitted, then no value is returned.

disk

This is the symbol used to return the amount of free space left on the specified disk. If this parameter is omitted, then no value is returned.

6.14A.4 Function Qualifiers

/NOPROMPT

The single optional qualifier /NOPROMPT causes the function to execute without interacting with the user. If /NOPROMPT is omitted and the threshold test fails, the user is told and asked whether processing should continue.

6.14A.5 Examples

Interactively:

```
<CHECK__DISKQUOTA "SMITH","USER1"
```

checks that user SMITH has at least 100 blocks of disk quota remaining on USER1.

6.14A.6 Special Features and Restrictions

The symbol OA$STATUS is set to 1 if the available space exceeds the specified threshold, or if the user has indicated when prompted that processing should continue.

6.15 CLEAR

6.15.1 Syntax

CLEAR [FORM]

6.15.2 Description

The CLEAR function clears the screen. You can call this function from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.15.3 Function Parameters

The single optional CLEAR parameter is the keyword FORM. FORM is primarily used by applications which do not use ALL-IN-1 to maintain the screen. An example is calling a nonintegrated editor like FMS. This changes the screen, but because FMS is not aware that the screen changed, when you return to ALL-IN-1 the previous form is not repainted. Using CLEAR FORM causes the original form to repaint when ALL-IN-1 is restored.

6.15.4 Function Qualifiers

None.

6.15.5 Examples

Interactively

```
<CLEAR
```

From Named Data:

Named Data

```
1  Name  ADD
FORM ADD/MODE=UPDATE\CLEAR\COMMAND APPLICATION
```

From the subprocess:

```
$ WRITE OAMAILBOX "OA CLEAR"
$ @DCLMAILBOX:
```

OR

```
$ WRITE OAMAILBOX "OA CLEAR\SCRIPT RUNTIME"
$ @DCLMAILBOX:
```

6.15.6 Special Features and Restrictions

When you call CLEAR interactively or by itself from Named Data, the function just repaints the screen. The most common application is a call from the subprocess to clear the screen prior to calling some application or function that does not automatically clear the screen itself.

/OVERLAY-qualified forms refresh the underlying form when they exit. This means that, although the underlying form does not fully repaint, broadcast messages, prompts, and other changes that may have been unrelated to the overlay are erased when the overlay exits. In contrast, if the current form is an overlay, calling CLEAR with the FORM keyword just refreshes that part of the screen that the current form overwrites.

6.16 CLOSE__PRIOR

6.16.1 Syntax

CLOSE__PRIOR

6.16.2 Description

The CLOSE__PRIOR function closes the current form (usually the last form displayed) and makes the previous form current. You usually use this function to close menus but you can also use it for any form .TYPE.

When post-processing forms or performing menu options, ALL-IN-1 looks for field or form terminators on the form that is displayed last. If a stack of functions calls a form, ALL-IN-1 looks for field or form terminators on this new form, unless you use the CLOSE__PRIOR function. For example, if Form B is called from Form A, functions falling after this call apply to Form B. To return control to Form A, you use the CLOSE__PRIOR function to close Form B. Then, all functions following the CLOSE__PRIOR call, affect Form A.

You can call CLOSE__PRIOR from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.16.3 Function Parameters

None.

6.16.4 Function Qualifiers

None.

6.16.5 Examples

In Named Data of form WPPARG1:

```
1 Name PL
FORM WPPARG2\.IF #CHOICE EQS "1" THEN .FX CLOSE__PRIOR__\OA$FLD__EXIT
```

WPPARG1 is a menu that calls itself as an argument form to gather print settings. Option PL calls another argument form, WPPARG2, to gather page layout information. If the user presses EXIT SCREEN on form WPPARG2, then local symbol #CHOICE is set to "0", the ".IF" test is not passed, and the user returns to WPPARG1. However, if the user presses RETURN to dispatch WPPARG2, then #CHOICE is set to "1", WPPARG1 is closed, and the user returns to the form that called WPPARG1.

In Named Data of form LPDOCSEL:

```
5 Name TITLE
/GET__SAVE=#TITLE/PUT__SAVE=#TITLE/RSE__RECOG=CAB$.TITLE

6 Name TITLE
WITH .FOLDER = FOLDER AND .TITLE = TITLE AND .DOCNUM EN DOCNUM;

7 Name TITLE
CLOSE__PRIOR\GET DOCNUM=OA$SEL__KEY:6:30
```

In field TITLE on form LPDOCSEL, CLOSE__PRIOR is called as a post-function from a validation expression. It closes the form that was current prior to LPDOCSEL. The /USE__FORM field qualifier then calls DOCRECOG to display the record collection.

6.17 COMMAND

6.17.1 Syntax

COMMAND com-file[/DCL_qual = arg][P1 . . . P8]

6.17.2 Description

The COMMAND function executes DIGITAL Command Language (DCL) procedures in the subprocess. You call this function from the following locations:

- In Named Data
- Interactively
- In templates
- In scripts
- Within validation expressions

See the VAX/VMS documentation set for detailed information on command procedures.

6.17.3 Function Parameters

com-file

The single, required parameter is the name of any command procedure. You must specify at least a file-name root: the file-type extension is optional and defaults to .COM when omitted. You can also specify a device and/or directory prefix.

P1 . . . P8

The parameters P1 through P8 are optional DCL parameters that pass to a command procedure. These are location-dependent constants: that is, the nth parameter is Pn. When COMMAND submits the command procedure to the subprocess, the DCL command interpreter associates the DCL symbol Pn with the value you assigned to the nth parameter. The command interpreter substitutes your value for Pn wherever it encounters Pn within the command procedure, consistent with the DCL syntax rules for symbol substitution.

Separate each DCL parameter with one or more blanks, and observe the following restrictions:

- If the first DCL parameter begins with a slash (/), enclose the parameter in quotes (") to distinguish it from the /OUTPUT qualifier, if any.
- If a DCL parameter contains embedded blanks or literal lowercase letters, enclose the parameter in quotes. For example:

  ```
  <COMMAND SPELLCOM "All"
  ```

- If a DCL parameter contains literal quotation marks, enclose the entire string in quotation marks and use a double set of quotes in the string. For example:

  ```
  <COMMAND INSPIRE "Never say ""never"""
  ```

- If you omit a parameter, put a pair of double quotes in its place. For example:

  ```
  <COMMAND SPELLCOM "ALL" "" "Nocheck"
  ```

6.17.4 Function Qualifiers

/OUTPUT = file-spec

The single, optional COMMAND qualifier is the DCL qualifier /OUTPUT. /OUTPUT requests that all output directed to the logical device SYS$OUTPUT be written to the file or device specified by file-spec.

You can optionally specify a device, directory, and file extension for file-spec. When you omit these, DCL assumes the default device and directory and a file extension of .LIS.

When specified, /OUTPUT must immediately follow the required com-file parameter. Otherwise, DCL interprets it as a parameter to pass to the command file.

NOTE: The /NEXT and /ARG qualifiers, which ALL-IN-1 recognizes in previous versions, are no longer valid COMMAND qualifiers.

6.17.5 Examples

In Named Data:

Named Data

```
1  Name  P
FORM PRTARG\IFEXIT\COMMAND DOCPRINT DOC
```

Menu option P displays form PRTARG, then executes DOCPRINT.COM with parameter P1 equal to DOC. If the user presses EXIT SCREEN the command procedure does not execute.

Named Data

```
1  Name  E
COMMAND INTERACT/OUTPUT=SYS$PRINT "EDIT" \ EDIT $EDITDOC
```

Menu option E first executes INTERACT.COM with parameter EDIT. DCL directs any output to the device specified by logical SYS$PRINT. When INTERACT exits, the EDIT function places the user in the editor with the file identified by the $EDITDOC symbol.

In a template:

```
<&OA COMMAND DOCRUNOFF /OUTPUT=DSR.LOG /RIGHT=5 INPUT.RNO>
```

This directive executes DOCRUNOFF.COM. DCL directs all output to the file DSR.LOG. Parameter P1 is "/RIGHT=5", P2 is "INPUT.RNO". DOCRUNOFF.COM follows:

```
$ ! Run the text formatting program RUNOFF
$ ! P1 is the right margin; P2 is the input file.
$ !
$ RUNOFF 'P1' 'P2'
$ !
$EXIT
```

File INPUT.MEM now contains the RUNOFF-formatted version of the input from INPUT.RNO; any error messages are stored in DSR.LOG.

Interactively:

```
<COMMAND DOCRUNOFF "/OUTPUT=OUTPUT.LIS" /RIGHT=5 INPUT.RNO
```

The quotes around /OUTPUT=OUTPUT.LIS now identify this as parameter P1 and not as the DCL COMMAND qualifier. COMMAND passes this parameter to the DCL command interpreter along with the other two parameters. DOCRUNOFF.COM follows:

```
$ ! Run the text formatting program RUNOFF with qualifiers
$ ! P1 and P2 and input file P3.
$ !
$ RUNOFF 'P1' 'P2' 'P3'
$ !
$EXIT
```

File OUTPUT.LIS now contains the RUNOFF-formatted output of the input from INPUT.RNO.

6.17.6 Special Features and Restrictions

There are several ways to create and manipulate ALL-IN-1 and DCL symbols in command procedures running in the subprocess.

One useful tool that allows you to trace the progress of a command procedure and the values assigned to symbols is the DCL SET VERIFY command. When you are running command procedures interactively, press Gold "(" to set DCL VERIFY Mode; press Gold ")" to set NOVERIFY.

Also, OA$LIB:V.COM is a simple command procedure that runs other command procedures in VERIFY Mode. Specify:

```
COMMAND V com-file [P2 . . . P8]
```

where com-file is the name of the command procedure to run in VERIFY Mode. The COMMAND function passes the com-file name as P1 to V.COM; V.COM in turn passes optional parameters P2 through P8 to this com-file as parameters P1 through P7. V.COM turns VERIFY Mode off when com-file exits.

The following restrictions apply to the COMMAND function:

- The COMMAND function no longer recognizes the /NEXT and /ARG qualifiers, which were valid in ALL-IN-1 Version 1. /ARG specified an argument form for a command procedure, and /NEXT specified a form to display after the command procedure exited. You can now implement these features by stacking FORM function calls.
- You cannot call the COMMAND function from the subprocess. To run a command procedure from the subprocess, use the standard DCL syntax:

  ```
  $ @com-file[/OUTPUT=file-spec [P1 . . . P8]
  ```

  where DCL interprets the @ symbol as a command-procedure execution instruction, the /OUTPUT is an optional qualifier, and P1 through P8 are optional parameters. When you are in the subprocess, you usually specify this @com-file statement within another command procedure: you can nest up to eight command procedures in this fashion.

  Unlike the COMMAND function, DCL assumes that a command procedure to be executed is in the default directory when you omit the directory prefix. However, you can use command procedure OA$LIB:FINDCOM in the subprocess to simulate the COMMAND-function search strategy.

  Instead of specifying @com-file, specify @OA$LIB:FINDCOM com-file. OA$LIB:FINDCOM treats its P1 parameter as a .COM file name and searches for it first in OAUSER, then in OA$LIB. When it finds the com-file, OA$LIB:FINDCOM executes it, passing its own P2 through P8 parameters to com-file as parameters P1 through P7.

  When you use the @com-file syntax with the interactive DCL function — for example, <DCL @com-file — you lose some DCL capabilities.
- Partial commands that result in system prompts do not work in any command procedure. Additionally, the DCL verb INQUIRE, which prompts for a symbol value, does not work in the subprocess and may cause other errors by hanging the subprocess. VMS does not support prompts when SYS$INPUT is directed to a mailbox, as is the case with the DCL and COMMAND functions.

- If you reset your directory with SET_DEFAULT, an ALL-IN-1 function, or the DCL command SET DEFAULT, you should change it back to what it was originally (usually OAUSER) before returning to the main process. Many functions and qualifiers in ALL-IN-1 assume that the default directory is the one identified in User Profile field DIRECT. The logical OAUSER is created when the subprocess is first initialized to facilitate this reset: simply enter the command SET DEFAULT OAUSER to reset the default ALL-IN-1 directory.
- Do not use the STOP or LOGOUT commands. They delete the subprocess.
- ALL-IN-1 does not open the subprocess and copy ALL-IN-1 permanent symbols to it, until the subprocess is needed. Before you use any ALL-IN-1 permanent symbols you must open the subprocess using the OA$SUB_OPEN command. Or, you can use the permanent symbols that you need by accessing them from the MAILBOX while in the command procedure.

6.18 COMPUTE

6.18.1 Syntax

COMPUTE [dest-sym = [dest-sym =] . . .]expression

6.18.2 Description

COMPUTE evaluates an arithmetic expression and optionally stores the result in one or more symbols. The expression may contain mathematical operators, numerals and symbols representing numerals. Both the destination symbols and the symbols in the expression may be either fields on the current form or symbol-table symbols.

You can call COMPUTE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the process
- In templates
- In scripts
- Within validation expressions

The precision of computed values is approximately 33 digits. Values range from .84 * 10(−4932) through .59 * 10(+4932).

6.18.3 Function Parameters

dest-symbol. . .

You optionally specify one or more destination symbols. The "=" is required. The dest-symbol parameters are ALL-IN-1 symbol-table symbols or fields on the current form that receive the value of the expression.

For each dest-symbol that is a permanent symbol, COMPUTE creates or updates (if the subprocess is open) a global DCL symbol and gives it the value of the expression and the same name (less the $ prefix) as dest-symbol.

When you omit dest-symbol, COMPUTE displays the value of the expression on line 24 of the screen, unless you have called COMPUTE from the subprocess. In that case, COMPUTE gives global DCL symbol RESULT the value of the expression.

expression

The second, required parameter is any meaningful combination of operators and operands. Table 6-1 gives the order of precedence for operators and operands. Use standard algebraic syntax to build an expression.

Table 6-1 COMPUTE Operators and Operands

Precedence	Operator	Function
1	(e1)	Subexpression
2	+ e2	Unary + (positive sign)
	− e2	Unary − (negative sign)
3	e1 ^ e2	Exponentiation
4	e1 * e2	Multiplication
	e1 / e2	Division
	e1 MOD e2	Remainder of e1/e2
5	e1 + e2	Addition
	e1 − e2	Subtraction

(continued)

Table 6-1 COMPUTE Operators and Operands (Cont.)

Precedence	Operator	Function
6	ABS(e1)	Absolute value
	ACOS(e1)	Arc cosine (e1 = cosine)
	ASIN(e1)	Arc sine (e1 = sine)
	ATAN(e1)	Arc tangent (e1 = tangent)
	ATN(e1)	Arc tangent (e1 = radians)
	COS(e1)	Cosine (e1 = radians)
	DEG(e1)	Radians to degrees
	FAC(e1)	Factorial
	FLOOR(e1)	Truncated integer
	INT(e1)	Nearest integer
	LOG(e1)	Common logarithm (base 10)
	LOGE(e1)	Natural log
	RAD(e1)	Degrees to radians
	SQRT(e1)	Square root
	SIN(e1)	Sine (e1 = radians)
	TAN(e1)	Tangent (e1 = radians)
	Operands	
	number	Numeric constant
	symbol	Numeric variable *
	PI	PI (3.14159265358979323846264338327 9)
	e	e (2.71828182845904523536028747135 2)

*Numeric variables may be symbol-table symbols or fields on the current form.

6.18.4 Function Qualifiers

None.

6.18.5 Examples

Valid expressions:

```
3*SQRT(9)

45 * ((9/2) + LOG($RATE/100))

0

50 MOD 3

(33.6783 * PI)^2
```

In Named Data:

Named Data

```
1  Name  C
COMPUTE ARG1 * ARG2 / SQRT(14) + 5 * $NUMERAL
```

Menu option C calls COMPUTE, which evaluates the expression and displays the result on line 24. ARG1 and ARG2 are fields on the menu, SQRT(14) evaluates the square root of 14, and $NUMERAL is a permanent symbol.

Named Data

```
1  Name  C
COMPUTE ANSWER = ARG1 * ARG2 / SQRT(14) + 5 * $NUMERAL
```

This follows the same computation as above, but the answer is stored in field ANSWER instead of displaying on line 24.

Interactively:

```
<COMPUTE #A + $B
```

This interactive call displays the sum of local symbol #A and permanent symbol $B on line 24.

```
<COMPUTE #ANSWER = ARG1 * ARG2 / SQRT(14) + 5 * $NUMERAL
```

COMPUTE evaluates the expression and stores the answer in local symbol #ANSWER.

```
<COMPUTE $ANSWER = 1 + 2
```

Permanent symbol $ANSWER now has a value of 3.

In a command procedure:

```
$ ! ARG1 and ARG2 are fields on the current form. ARG3 is a DCL
$ ! symbol that has the same value as permanent symbol
$ ! $NUMERAL. DCL substitutes a value for ARG3 before WRITE.
$ !
$ WRITE OAMAILBOX "OA COMPUTE ARG1 * ARG2 / SQRT(14) + 5-"
$ WRITE OAMAILBOX "OA * ''ARG3'"
$ @DCLMAILBOX:
$ !
$ ! The result of the computation is returned through DCLMAILBOX,
$ ! where it is stored in symbol RESULT.
$ !
$ WRITE SYS$OUTPUT "The result is ''RESULT'"
```

Note the continuation character ("-") at the end of the first WRITE to OAMAILBOX in this example. ALL-IN-1 processes this before calling COMPUTE, and it is *not* interpreted as a minus sign. ALL-IN-1 dispatches the COMPUTE function with the following expression:

ARG1 * ARG2 / SQRT(14) + 5 * . . .

```
$ ! An expression is evaluated and the result is
$ ! assigned to the permanent symbol $DELTA.
$ WRITE OAMAILBOX "OA COMPUTE $DELTA = OA$DATE_NBS:8-$ORIG_DATE"
$ !
$ ! OA$DATE_NBS:8 represents the first eight numerals
$ ! of the current date in NBS format.
$ !
$ @DCLMAILBOX:
$ !
$ ! DCL symbol DELTA contains the difference:
$ !
$ WRITE SYS$OUTPUT "The result is ''DELTA'"
```

6.18.6 Special Features and Restrictions

COMPUTE and the /COMPUTE field qualifier calculate to the least significant digit and round off to the nearest decimal place, which is identified by the DECIMAL function. For more discussion see the DECIMAL function.

6.19 CONTROL_C

6.19.1 Syntax

CONTROL_C { YES | NO }

6.19.2 Description

The CONTROL_C function enables/disables the Control-C trap processing for ALL-IN-1's main process. When the Control-C handler is enabled, ALL-IN-1 processes any CTRL/Cs the user types. If the Control-C handler is disabled, ALL-IN-1 passes any CTRL/Cs the user types to the application or layered product that is currently active.

You can call CONTROL_C from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.19.3 Function Parameters

{ YES | NO }

The single, required parameter is either YES or NO. The parameter YES enables the Control-C handler, while the parameter NO disables it.

6.20 COPY

6.20.1 Syntax

COPY infile_symb outfile_symb

6.20.2 Description

The COPY function copies an RMS file or pipeline-source output to another RMS file, or to a special destination symbol.

You call COPY from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.20.3 Function Parameters

infile_symb

The first COPY parameter is an ALL-IN-1 symbol representing a single file to copy. You can specify any valid RMS file specification.

outfile_symb

The second COPY parameter is an ALL-IN-1 symbol representing the file to copy to. COPY creates the file if it does not already exist. You can specify any valid RMS file specification. Note that if the file extension is a recognised file type, COPY automatically converts the file to the format of that file type.

In Pipeline Mode, COPY copies the output generated by the pipeline source-function to outfile_symb. The syntax is:

```
COPY "<source_function" outfile_symb
```

6.20.4 Function Qualifiers

None.

6.20.5 Examples

Interactively:

```
<COPY "OLDDATA.TMP" "NEWDATA.TMP"
```

This function call copies OLDDATA.TMP to NEWDATA.TMP.

```
<COPY "@DOC-NAME-SYN" "NEWDATA.TMP"
```

This function call copies the document and all of its attachments specified by DOC-NAME-SYN to NEWDATA.TMP.

6.20.6 Special Features and Restrictions

When you omit device and directory prefixes for either the infile or the outfile, COPY assumes that the file is in the user's current default directory (generally OAUSER). Other than this, COPY does not assume any default file-creation information, nor does it take default information for the outfile from the infile specification (as the DCL COPY command does). The outfile is created exactly as specified.

6.21 CREATE

6.21.1 Syntax

CREATE entry-form

6.21.2 Description

The CREATE function creates an indexed RMS data file from an entry form. The .FILE directive in Named Data of the entry form provides the name of the data file to create, as well as instructions on how to create the file.

You can call CREATE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.21.3 Function Parameters

entry-form

The single, required parameter is the name of the standard entry form that contains a .FILE directive. CREATE uses the FMS field attributes and layouts of the specified form to build the physical records, and it extracts information on how to index the file from the form's Named Data.

/KEYS and other form qualifiers are ignored during the file creation. The Named Data .FILE directive alone is used to create the file. If an FDL specification is present on the .FILE line, it is used only during CREATE.

6.21.4 Function Qualifiers

None.

6.21.5 Examples

In Named Data:

Named Data

```
1 Name F
CREATE NEWDB\FORM NEWDB/MODE=ADD
```

This option creates the file specified by entry form NEWDB and then displays the form in the ADD Mode.

Interactively:

```
<CREATE NEWDB
```

CREATE creates the file specified by entry form NEWDB.

6.21.6 Special Features and Restrictions

When the form specified as the CREATE parameter describes an existing data file, CREATE creates a new version of the file.

If you have altered a field on an entry form with the FMS forms editor and the form no longer describes the file accurately, you must create a new file. You can not access records in the old file through the altered entry form, and you must either transfer or delete them. When you have changed the size of one or more fields or added one or more fields, use the function OA$CNV_CONVERT to copy records from an outdated file to a new version. When you rename fields or wish to delete fields, call the FOR function.

6.22 CXP

6.22.1 Syntax

CXP subfunction [/subfunction_qual[/subfunction_qual]...]

6.22.2 Description

CXP, the Communications function, accesses the ALL-IN-1 Communications Control application. This application provides tools for connecting the VAX on which ALL-IN-1 is running to some other computer or network of computers that supports ASCII terminals. The connection may be through modems or hardwired, and you can invoke Communications Control documents to automate the process.

You can call CXP from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In scripts

6.22.3 Subfunction Descriptions

Chapter 12 describes the CXP subfunctions.

6.23 DATE__CONVERT

6.23.1 Syntax

DATE__CONVERT dest__sym,source__sym, format__key

6.23.2 Description

The DATE__CONVERT function converts dates from one format to another and stores the converted date in an ALL-IN-1 symbol. You call this function from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.23.3 Function Parameters

dest__sym

The first DATE__CONVERT parameter is an ALL-IN-1 symbol-table symbol or a field on the current form that receives the converted value. If dest__sym is a permanent symbol, ALL-IN-1 also creates (or updates) a global DCL symbol in the subprocess with the same value and the same name (less the "$" prefix) as dest__sym.

source__sym

The second DATE__CONVERT parameter is any ALL-IN-1 symbol or literal string that represents the date to be converted. DATE__CONVERT recognizes the following source__sym formats as valid:

- mm/dd/yy — Special symbol OA$DATE is in this format.
- dd-mmm-yyyy — Special symbol OA$DATE__FULL is in this format.

- mm/dd/yy [hh:mm]
- dd-mmm-yyyy [hh:mm]
- yyyymmddhhmmss — NBS format; special symbol OA$DATE_NBS is in this format.

A format is only meaningful when specified in conjunction with the proper format key, as illustrated in Table 6-2.

format_key

The third DATE_CONVERT parameter is a numeral from 1 to 14 that identifies the desired format of dest_sym. Table 6-2 illustrates the source_sym formats that you can specify with a given format_key.

Table 6-2 DATE_CONVERT Parameters

If you specify format key:	DATE_CONVERT expects source_sym in the format:	and will convert dest_sym to the format:
1	mm/dd/yy	mm/dd/yy
2	or	dd-mmm-yyyy
3	dd-mmm-yyyy	dd-mmm-yyyy hh:mm
4	or	hh:mm
5	mm/dd/yy [hh:mm]	day of the week
6	or	month
7	dd-mmm-yyyy [hh:mm]	yyyymmddhhmmssmm (NBS)
8		mm/dd/yy
9		dd-mmm-yyyy
10		dd-mmm-yyyy hh:mm
11	yyyymmddhhmmss (NBS)	hh:mm
12		day of the week
13		month
14		yyyymmddhhmmss (NBS)

6.23.4 Function Qualifiers

None.

6.23.5 Examples

Interactively:

```
<DATE_CONVERT $TEMP_DATE, OA$DATE, 2
```

DATE_CONVERT converts the current system date, stored in special symbol OA$DATE in mm/dd/yy format, to dd-mmm-yyyy format and stores it in permanent symbol $TEMP_DATE.

In Named Data:

Named Data

```
2 Name .TYPE
/POST = "DATE_CONVERT AI_SCHED_DATE_NBS,AI_SCHED_DATE,7"
```

DATE_CONVERT converts the value of field AI_SCHED_DATE, which is the date in dd-mmm-yyyy format, to NBS format and stores the value in field AI_SCHED_DATE_NBS.

In a command procedure:

```
$ WRITE OAMAILBOX "OA DATE_CONVERT $HOLD_DATE,''p1','p2'"
$ @DCLMAILBOX:
```

This call converts the value of DCL parameter p1 into the format identified by parameter p2. DATE_CONVERT stores the result in permanent symbol $HOLD_DATE. Global DCL symbol "HOLD_DATE" also contains the converted date.

6.24 DCL

6.24.1 Syntax

DCL [DCL-command-string]

6.24.2 Description

The DCL function provides single-line, interactive access to the subprocess at the DCL level. This function may be called in the following ways:

- From Named Data
- Interactively, by entering <DCL or $ or @
- From the default editor, by entering DCL as a Line-Mode command or pressing Gold $ in change Mode
- In templates
- In scripts
- Within validation expressions

In addition, Gold $ puts you at DCL level when pressed in any field because this key sequence is associated with the DCL function in the Named Data of form DEFAULT. Gold $ also works from the EDT and WPS style editors.

6.24.3 Function Parameters

DCL-command-string

The DCL Command Language Interpreter (CLI) processes the optional DCL command string as if it were a command entered at a $ prompt. You are placed at another $ prompt after DCL executes the command.

When you call DCL with no parameters, the function places you in the subprocess at a $ prompt. You remain in interactive DCL Mode until you press RETURN with no command on line.

See the VAX/VMS documentation set for a list of DCL commands and their qualifiers.

6.24.4 Function Qualifiers

There are no ALL-IN-1 qualifiers for the DCL function, but many DCL commands take qualifiers. The VAX/VMS documentation set lists these qualifiers.

6.24.5 Examples

Interactively:

```
<DCL SHOW SYSTEM

$ SHOW SYSTEM

<Gold> $ SHOW SYSTEM
```

These three examples show the current status of the VMS operating system: the first two methods are valid from a menu choice field, and the third method, pressing Gold $, is valid from any enterable field on any form.

6.24.6 Special Features and Restrictions

To use the DCL command, or Gold $ interactively, you must have the profile privilege PRVDCL.

You can send DCL command lines to the subprocess with the OA$DCL special GET destination. Unlike the DCL function, OA$DCL returns control to the point of call after DCL executes the command. Also, OA$DCL translates ALL-IN-1 symbols or symbol lists. The syntax is:

```
GET OA$DCL = command-line-symbol
```

where command-line-symbol is any combination of ALL-IN-1 symbols.

You cannot call the DCL function from the subprocess, where it would be superfluous. Other restrictions follow:

- Complete or partial commands that result in system prompts do not work. For example, entering the command COPY at DCL level outside of ALL-IN-1 results in $_To: and $_From: prompts, but the command results in an error message when entered as an interactive DCL command while running ALL-IN-1:

  ```
  $ COPY
  ```

  OR

  ```
  <DCL COPY
  ```

  results in

  ```
  Insufficient parameters
  ```

  Similarly, the DCL verb INQUIRE, which prompts for a symbol value, does not work. VMS does not support prompts when SYS$INPUT is directed to a mailbox, as is the case with the DCL and COMMAND functions.

- You cannot specify continuation lines — the ALL-IN-1 command dispatcher does not recognize the DCL continuation character in the context of a DCL function call. Interactive DCL commands must fit on a single line of no longer than 255 characters.

- If, in a subprocess, you reset your directory with the ALL-IN-1 SET_DEFAULT function or the DCL command SET DEFAULT, you should change it back to what it was originally (usually OAUSER) before returning to the main process. To reset the default ALL-IN-1 directory, enter the command SET DEFAULT OAUSER. If you do not do this, functions and qualifiers that try to reference work files in the user's default directory will be unable to find them.

- Do not use the STOP or LOGOUT commands. They delete the subprocess.

6.25 DECIMAL

6.25.1 Syntax

DECIMAL places

6.25.2 Description

Use the DECIMAL function in conjunction with the COMPUTE function and the /COMPUTE field qualifier to evaluate a mathematical expression to a specific number of decimal places.

6.25.3 Function Parameters

places

The single, required parameter is the number of decimal places desired. This parameter may be:

- An integer from 0 to 22
- F for floating-point
- I for rounded integer

The default is F. A floating-point display is a two-part numeric expression used to represent very large or very small numbers. The display consists of a base number (usually between 0 and 1.0) followed by an expression with the syntax:

E<sign>n

where

E	is shorthand notation for 10^, or 10 raised to the power of
<sign>	is + or −.
n	is the exponent, an integer

The range of the exponent is approximately −4932 to +4932. The actual quantity represented by a floating-point display is thus

base number * 10 ^ exponent

For example:

1.00046E+27

is the floating-point equivalent of 1.00046 * 10^27.

Specifying DECIMAL I causes subsequent COMPUTE and /COMPUTE to round their results to the nearest integer.

6.25.4 Function Qualifiers

None.

6.25.5 Examples

In Named Data:

Named Data

```
1 Name C
DECIMAL 2\COMPUTE SQRT(14)

2 Name EXP
DECIMAL F\GET OA$FUNCTION="COMPUTE 2^" SYMBOL
```

Menu option C computes the square root of 14 to 2 decimal places. Option EXP computes 2 to the power that is represented by SYMBOL and formats the answer in floating-point format. Both answers appear on line 24 of the screen.

6.25.6 Special Features and Restrictions

The P option on form OA$DESK sets the number of decimal places independently of the DECIMAL function. ALL-IN-1 saves the number of places you specify on the OA$DESK form from session to session independently of the number of places you set with DECIMAL. Whenever you call the DECIMAL function, ALL-IN-1 calculates all subsequent computations (other than those on form OA$DESK) to that number of places, until you call the function again with a new parameter.

6.26 DELETE__FILE

6.26.1 Syntax

DELETE__FILE RMS-file[,RMS-file ...]

6.26.2 Description

The DELETE__FILE function deletes all versions of the specified RMS file or files. The syntax and results of this function are similar to those of the VMS DELETE command, but DELETE__FILE does not require version numbers.

You call DELETE__FILE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.26.3 Function Parameters

RMS-file...

You can specify one or more RMS file names, separated by commas. DELETE__FILE recognizes full RMS file specifications and translates logical names that have been defined in the context of the main process, but it does not recognize wild cards in the file specification.

When you omit the directory prefix(es), DELETE__FILE assumes the file(s) are in your default ALL-IN-1 directory, OAUSER, rather than the current directory, if the two are different.

6.26.4 Function Qualifiers

None.

6.26.5 Examples

In Named Data:

Named Data

```
1 Name DT
DELETE__FILE [.TEMP]TEST.TMP\DISPLAY Temporary file deleted\FORCE
```

When the user selects menu option DT, DELETE__FILE deletes the file TEST.TMP in subdirectory .TEMP, and DISPLAY displays the specified message.

Interactively:

```
<DELETE__FILE THISFILE.TXT,SYS$LOGIN:THATFILE.TMP
```

THISFILE.TXT in OAUSER and THATFILE.TMP in SYS$LOGIN are deleted interactively.

6.26.6 Special Features and Restrictions

Like PURGE__FILE, DELETE__FILE acts in the main process and is thus subject to the privileges, default protections, logical name translation, and other characteristics of the main process as opposed to those of the subprocess. Thus, if you alter privileges in the subprocess to gain delete access to a protected file, you cannot then delete the file with DELETE__FILE; you must use the DCL DELETE command. For example, if file DELETEME.TXT had the following protection:

```
(SYSTEM:RWED,OWNER:RWED,GROUP:RE,WORLD)
```

and you were not the owner of this file, then the following sequence from the subprocess would *not* work:

```
$ SET PROC/PRIV__=SYSPRV
$ WRITE OAMAILBOX "OA DELETE__FILE DELETEME.TXT"
```

Similarly, DELETE__FILE does not recognize logical names that have been assigned only for the subprocess.

6.27 DESTINATION

6.27.1 Syntax

DESTINATION data_set

6.27.2 Description

The DESTINATION function identifies a text data set as the destination for the output of the next ALL-IN-1 function call that generates output. You call this function from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.27.3 Function Parameters

output-file

The single, required DESTINATION parameter is the name of a text data set.

6.27.4 Function Qualifiers

None.

6.27.5 Special Features and Restrictions

Chapter 10 covers text data sets in more detail.

6.28 DISPLAY

6.28.1 Syntax

DISPLAY text

6.28.2 Description

The DISPLAY function displays a one-line, boldfaced message on line 24 of the screen. ALL-IN-1 continues processing immediately after the message displays.

You call DISPLAY from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.28.3 Function Parameters

text

The single, required parameter for the DISPLAY function is the text string to display. The string may contain backslashes (\), but to distinguish them from stacked-function indicators, double the backslashes. That is, enter \\ to display \. If the text is longer than 80 characters, DISPLAY truncates it to the current size of the screen.

You can call DISPLAY with a symbol or symbol list as the text parameter using a special output destination, OA$DISPLAY.

6.28.4 Function Qualifiers

None.

6.28.5 Examples

In Named Data:

Named Data

```
1 Name ?
DISPLAY Press Gold H (PF1 on the keypad and then H) for help.
```

Typing a question mark on this form displays the following message on line 24:

```
Press Gold H (PF1 on the keypad and then H) for Help.
```

In a command procedure:

```
$ ! Display a message.
$ !
$LOOP:
$ WRITE OAMAILBOX "OA DISPLAY Processing 'THIS_ONE'...\FORCE"
$ @DCLMAILBOX:
$ !
$ ! Continue loop...
```

In this example, the command procedure displays the text "Processing" and the contents of the DCL symbol THIS_ONE every loop-through. The DISPLAY function keeps the user posted on the progress of the procedure.

6.28.6 Special Features and Restrictions

The following restrictions apply to DISPLAY function calls:

- The display text is a literal string, and when quotes are present, DISPLAY includes them in the message. If the text is longer than 80 characters, DISPLAY truncates it to the current length of the screen.

- When a DISPLAY string contains backslashes (\), which separate stacked functions, type two backslashes to display the message properly. For example:

  <DISPLAY Double up on backslashes (\\)

  would display

  ```
  Double up on backslashes (\)
  ```

  In command procedures, the display string should not contain apostrophes (single quotes), which are interpreted by DCL as symbol-substitution characters. To indicate quoted material, use normal quotes, and double them so DCL processes the string properly. For example, the following function call:

  ```
  $ THIS ONE := "Chapter 6"
  $ WRITE OAMAILBOX "OA DISPLAY Processing """'THIS_ONE'"""
  $ @DCLMAILBOX:
  ```

  results in the following message:

  ```
  Processing "Chapter 6"
  ```

- To display the values of symbol-table symbols, use GET=OA$DISPLAY. For example:

  Named Data

  ```
  1 Name ?
  GET OA$DISPLAY = OA$USER ", Do you need help?"

  2 Name ?
  PROFIL.TITLE[OA$USER] " Press Gold H."
  ```

  OA$DISPLAY is a special output destination used with the GET function to expand the DISPLAY capability.

- ALL-IN-1 uses the Message Facility (Chapter 14) to output the text you specify with DISPLAY. This has the following effects:

  - ALL-IN-1 outputs DISPLAY messages through an informational message indexed by the keyword LASTLINE. That is, when you call the DISPLAY function, you are actually signaling a message. If the user presses Gold W after the message displays, the message buffer shows:

    ```
    %OA-I-LASTLINE, your text
    ```

- Also, if the user presses Gold H immediately after a DISPLAY function call, HELP builds its keyword list internally as ERRORS LASTLINE. Since LASTLINE changes with every DISPLAY function call (as well as with other function calls), the help library contains the following direct reference to symbol OA$PROMPT_HELP:

  ```
  2 LASTLINE
  .br
  =OA$PROMPT_HELP
  ```

  Load this symbol with a keyword list every time you want to provide help on a DISPLAY message.

See also the PROMPT and ERROR functions in this chapter, and the Message Facility in Chapter 14.

6.29 DO

6.29.1 Syntax

DO script-file-name

6.29.2 Description

DO is a variation of the SCRIPT function. DO executes scripts in DO mode, a Script-Facility Mode in which the user satisfies ALL-IN-1 requests for input. In contrast, SCRIPT-invoked scripts, which run in SCRIPT Mode, provide (or control) all answers to ALL-IN-1 requests for input.

You call DO from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

NOTE: This is not the DO action-keyword that is specified in FOR function calls.

6.29.3 Function Parameters

script-file-name

The single, required parameter is the name of an RMS ASCII file that contains script commands. DO recognizes full device and directory specifications. When you omit device and directory prefixes, DO searches for the script file in the same directories and in the same order that the COMMAND function searches for a .COM file. The current default directory is searched first and then OA$LIB.

DO files contain directives and function calls. Directives have a period in the first column. DO treats all other lines as function calls.

Chapter 15 provides a full listing of the directives that are valid in DO-invoked scripts.

6.29.4 Function Qualifiers

None.

6.29.5 Examples

In Named Data:

Named Data

```
1 Name .TYPE
MENU/CHOICE = CHOICE/PRE = 'DO EXAMPLE'
```

Before this menu displays, the /PRE form qualifier calls DO to execute EXAMPLE.SCP.

Interactively:

```
<DO HARDCOPY
```

DO invokes HARDCOPY.SCP.

In a script:

```
!
! Call script NEXTSTEP.SCP in DO Mode if the status is true
!
.IF OA$STATUS THEN .FX DO NEXTSTEP
```

6.29.6 Special Features and Restrictions

Any SCRIPT-Mode feature that provides input to ALL-IN-1 is either invalid or treated differently in DO-invoked scripts. Specifically, DO scripts treat text lines as function calls in DO scripts. You can not include the .PROCESS directive in a DO script.

6.30 DTR

6.30.1 Syntax

DTR [dtr-command(s)]

6.30.2 Description

The DTR function invokes DATATRIEVE when callable DATATRIEVE is linked with ALL-IN-1. You may use DATATRIEVE interactively, or enter a command or series of commands and then automatically return to ALL-IN-1. Once the DTR function call establishes a link, you can access DATATRIEVE variables, field names and other data elements as ALL-IN-1 symbols.

You can call DTR from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.30.3 Function Parameters

dtr-command(s)

The single, optional DTR parameter is any valid DATATRIEVE command or series of commands. The VAX DATATRIEVE documentation set lists the DATATRIEVE commands.

When you call DTR with no parameters, the function acts like the DCL function: you are placed in interactive DATATRIEVE at the DTR› prompt, and you exit by pressing RETURN with no command. When you call DTR with parameters, the function passes those parameters to DATATRIEVE, and you do not enter an interactive mode.

When you invoke DATATRIEVE with the DTR function, ALL-IN-1 creates special symbols based on the values of any normal DATATRIEVE data element that you have selected, for example, symbols, fields, and other data elements. ALL-IN-1 creates the special symbols by prefixing the DATATRIEVE name of the element with DTR$. These symbols are then available to the ALL-IN-1 GET function. The VAX DATATRIEVE documentation set covers the DATATRIEVE SELECT command and the concept of selecting an element.

6.30.4 Function Qualifiers

None.

6.30.5 Examples

In Named Data:

Named Data

```
1 Name SD
DTR SET DICTIONARY CDD$TOP
```

Menu option SD sets the common data dictionary path to the top-level CDD$TOP.

Interactively:

```
<DTR
```

Enter interactive DATATRIEVE.

```
<DTR SET DICTIONARY CDD$TOP; READY DOCDB SHARED WRITE
```

The default dictionary is set to CDD$TOP and domain DOCDB is readied for shared write access. Control returns to the menu.

In a command procedure:

- Calling a DATATRIEVE procedure from the subprocess and passing a parameter to it:

```
$ ! Set up local DTR symbol for use in DTR procedure.
$ !
$ PAR1 :== "FIRST VALUE"
$ WRITE OAMAILBOX "OA DTR DECLARE PASS1 PIC X(30) -"
$ WRITE OAMAILBOX "OA DEFAULT VALUE IS ""'PAR1'""."
$ @DCLMAILBOX:
$ !
$ ! Call DTR procedure
$ !
$ WRITE OAMAILBOX "OA DTR :PROC1"
$ @DCLMAILBOX:
```

- Calling DATATRIEVE to put some records into a file to be printed:

```
$ WRITE OAMAILBOX "OA DTR PRINT ALL ON TEMP.LIS"
$ @DCLMAILBOX:
$ @OAUSER:DTRPRINT "TEMP.LIS"
```

- Selecting a DTR record and saving a field from the record through a DTR$ special symbol:

```
$ WRITE OAMAILBOX "OA DTR FIND DOCDB WITH CATEGO=""'P1'"" -"
$ WRITE OAMAILBOX "OA AND FILNAM CONT "".OUT"";SELECT"
$ @DCLMAILBOX:
$ WRITE OAMAILBOX "OA GET $OUTLINE__NAME=DTR$DOCNAM"
$ @DCLMAILBOX:
```

As a pre-function on a form:

```
/PRE='GET OA$DTR = "SET DICTIONARY CDD$TOP"
```

6.30.6 Special Features and Restrictions

DTR invokes DATATRIEVE through the main process, not the subprocess. Thus, any logicals you specify must be defined in the context of the main process. Also, when you call DTR and enter DATATRIEVE commands that alter the status of the Common Data Dictionary — ready a domain, establish a collection, select a record, and so on — the CDD remains in the state in which you left it until you exit from ALL-IN-1. The following sequence of commands illustrates this feature.

1 DATATRIEVE command is entered from a menu:

```
<DTR READY DOCDB AS DOCUMENTS SHARED WRITE
```

Control returns to the menu choice field.

2 Read the next mail message:

```
EM RN
```

3 After next message is read, return to DATATRIEVE:

```
<DTR SHOW READY
Ready domains:
   DOCDB: Domain, RMS Indexed, shared write
          (CDD$TOP.DOCDB)
No loaded tables.

Press RETURN to continue
```

The DATATRIEVE command FINISH may be used to release any resources used by READY domains, and so on.

6.31 DUMP__CACHE

6.31.1 Syntax

DUMP__CACHE [name]

6.31.2 Description

ALL-IN-1 keeps track of recently or frequently accessed data sets and caches them, or keeps them open, to improve performance. The DUMP__CACHE function flushes all data sets that ALL-IN-1 has cached.

You can call DUMP__CACHE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.31.3 Function Parameters

The single, optional parameter is the name of a single entity which you wish to dump from the cache.

For example:

DUMP__CACHE PROFIL

ensures that only the user profile file is closed. There is no error if PROFIL is already closed.

6.31.4 Function Qualifiers

None.

6.31.5 Examples

Interactively:

```
<DUMP_CACHE
```

ALL-IN-1 flushes all unused data sets.

In Named Data:

Named Data:

```
1 Name SW
OA$FLO_CLOSE OLDLIB \ DUMP_CACHE \

2 Name SW
OA$FLO_OPEN NEWLIB \ DISPLAY New form library opened\FORCE
```

Option SW on this menu executes a list of five functions that close one form library, flush all unused data sets, open another form library, and display a message explaining what has happened.

6.31.6 Special Features and Restrictions

The NEWLIB function calls DUMP_CACHE automatically before NEWLIB opens new versions of form libraries.

ALL-IN-1 keeps a number of data sets permanently open: File Cabinet data sets, editor-related data sets, CALENDAR data sets, and others. These open data sets are analogous to memory resident forms: there is no way to modify them while ALL-IN-1 is running.

To view all open data sets and forms, call the OA$DSA_CACHE_STATUS function.

You usually call DUMP_CACHE in conjunction with the OA$FLO_OPEN and OA$FLO_CLOSE functions, as illustrated in the last example in the preceding section. Those functions are covered elsewhere in this chapter.

6.32 EDIT

6.32.1 Syntax

EDIT [file-sym], [init-file], [purge-count] [,editor-type]

6.32.2 Description

Call the EDIT function to edit an existing file or create a new one. By default, EDIT reads the User Profile Editor Style field, EDITOR, to determine which editor to call.

You call EDIT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.32.3 Function Parameters

All EDIT parameters are positional, and you must add placekeeping commas if you omit a parameter.

file-sym

The file-sym parameter is any ALL-IN-1 symbol that represents an RMS file name. This parameter is optional; if you omit it, file-sym defaults to the value of special symbol OA$EDIT__FILE.

EDIT stores the name of the most recently edited file in OA$EDIT__FILE. You can also modify this updatable special symbol to cause EDIT to process another file.

When specified, the file-symbol parameter may be either a quoted string:

```
<EDIT "THISDOC.TXT"
$ WRITE OAMAILBOX "OA EDIT 'ANYDOC.TXT'"
$ @DCLMAILBOX:
```

or some other valid symbol reference to a file name:

```
EDIT #CURDOC

<EDIT DOCDB.FILENAME['STATUS REPORT']
```

init-file

The optional init-file parameter is an ALL-IN-1 symbol that represents an editor initialization file for the default ALL-IN-1 editor. This parameter is optional and defaults to the value of OA$EDIT_INI_FILE if you omit it.

EDIT saves the value of the most recently specified initialization file in OA$EDIT_INI_FILE. You can modify this updatable special symbol to cause EDIT to invoke another start-up file.

The init-file parameter is only meaningful for those editors, such as the default ALL-IN-1 editor, that allow you to specify a start-up file. EDIT ignores this parameter in all other cases. For the default editor, the default init-file type is .EDT. EDIT appends this extension if you only specify a file-name root.

purge-count

The optional purge-count parameter is a symbol that specifies the number of copies of the document to keep after you exit the editor. This parameter defaults to the value of special symbol OA$EDIT_PURGE_COUNT if you omit it.

EDIT saves the last specified purge count in OA$EDIT PURGE_COUNT. As with the other editor-related special symbols, you can update this symbol at will.

For example:

```
<EDIT'NEWFILE.TXT',,3
```

would purge all but three copies of NEWFILE.TXT after you exit the editor. Note the extra placekeeping comma that differentiates the purge-count parameter from the initialization-file parameter.

editor-type

The fourth optional parameter is an ALL-IN-1 symbol representing the name of the editor. When you omit editor-type, EDIT attempts to invoke the editor type identified in special symbol OA$EDITOR. If that symbol is null, EDIT invokes the editor identified in the User Profile EDITOR field (a required field), and then saves that value in OA$EDITOR. As with other editor-related special symbols, you can update OA$EDITOR at will.

For example:

```
<EDIT $THISDOC,,,'WPSPLUS'
```

6.32.4 Function Qualifiers

None.

6.32.5 Special Features and Restrictions

EDIT examines the file-type extension of the file-symbol parameter to determine the file's internal structure and then converts the file, if necessary, to allow the selected editor to process the file. Table 10-1 covers file-type extensions and their related structures.

6.33 EDT

6.33.1 Syntax

```
EDT form-name
```

6.33.2 Description

Use the EDT function to display a form while the user is in the default editor and to cause ALL-IN-1 to direct output from certain function calls to the editor's paste buffer. The functions whose output is redirected by EDT are the same functions that are affected by the DESTINATION function:

- SEARCH
- DTR
- GET
- COMPUTE

The EDT-specified form is usually a menu.

6.33.3 Function Parameters

form-name

The single, required parameter is the name of an ALL-IN-1 form to display. Calling

EDT form

is equivalent to calling

```
DESTINATION OA$EDIT_BUFFER\FORM form
```

6.33.4 Function Qualifiers

None.

6.33.5 Special Features and Restrictions

As mentioned previously, calling the EDT function is equivalent to calling the DESTINATION function followed by the form function. Thus, if you press Gold M and call SEARCH, DTR, GET, or COMPUTE while form EDTMEN is active, ALL-IN-1 stores the results of that function call in the paste buffer. Thus, when you return to the editor, you can include those results into the current document with one key sequence.

See also the INCLUDE function, which allows you to include files into the editor or into images while running in the subprocess.

6.34 ERROR

6.34.1 Syntax

ERROR [text]

6.34.2 Description

Call the ERROR function from command procedures running in the subprocess to inform ALL-IN-1 that it should not continue processing stacked functions. ERROR is similar to the IFEXIT function. IFEXIT causes ALL-IN-1 to ignore the functions that follow it in a list if the user presses EXIT SCREEN; ERROR has the same effect when you are calling a list from the subprocess. See also the IFEXIT function in this chapter.

ERROR is only meaningful when called from the subprocess.

6.34.3 Function Parameters

text

The single, optional ERROR parameter is an error message that displays on line 24 of the terminal when you call the function. Like the DISPLAY parameter, this message is a literal text string: ERROR makes no symbol substitution, and when quotes are present, ERROR includes them in the message. ERROR truncates the text to 80 characters if it is longer.

6.34.4 Function Qualifiers

None.

6.34.5 Examples

Assume that the Named Data of menu FC1 contains the following list of function calls associated with option RFD:

```
7 Name RFD
COM WPCHKDOC $WPDOC\FORM WPRFAG\IFEX\DISP Refiling document . . .

8 Name RFD
\FORCE\GET $CURDOC=$WPDOC\COMMAND FCDOCFIL REFILE\FORM
```

and command procedure WPCHKDOC.COM contains the following lines:

```
$ !
$ ! The following symbol is used in this procedure
$ !
$ ! P1 - contains the ALL-IN-1 Symbol name whose contents are
$ !      to be checked
$ !
$  WRITE OAMAILBOX "OA GET "P1'"
$  @DCLMAILBOX:
$  IF RESULT .NES. "" THEN EXIT
$
$  WRITE OAMAILBOX "OA ERROR No current document selected"
$  @DCLMAILBOX:
```

WPCHKDOC.COM calls ERROR through OAMAILBOX whenever $WPDOC does not contain a value, and ALL-IN-1 ignores the functions that follow the COMMAND function in Named Data.

6.34.6 Special Features and Restrictions

Although ERROR only recognizes a string literal as a parameter, you can use the GET OA$FUNCTION special output destination to call this function with a symbol or list of symbols.

ERROR displays a specified message if you call it interactively or from Named Data, but the DISPLAY function is the recommended way to put information other than error messages on line 24 of the screen.

6.35 EXIT

6.35.1 Syntax

EXIT

6.35.2 Description

The EXIT function closes all open files, stops execution of the ALL-IN-1 image, and exits to DCL command level.

You can call EXIT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.35.3 Function Parameters

None.

6.35.4 Function Qualifiers

None.

6.35.5 Examples

In Named Data:

Named Data

```
1 Name EX
EXIT
```

Menu option EX calls the EXIT function and puts the user at DCL command level.

Interactively:

```
<EXIT
```

6.35.6 Special Features and Restrictions

A line in OAINI.COM assigns the DCL synonym LOGOUT to the EXIT function call. OAINI.COM is the command procedure, in either OA$LIB or OAUSER, that runs in the subprocess. ALL-IN-1 creates the subprocess. Thus, entering LO at interactive DCL level in the subprocess logs you out of ALL-IN-1.

See also the BYE function.

6.36 FHELP

6.36.1 Syntax

FHELP

6.36.2 Description

The FHELP function lists all callable functions: control functions, user-defined functions, and internal functions.

You can call FHELP from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- From within validation expressions

6.36.3 Function Parameters

None.

6.36.4 Function Qualifiers

None.

6.36.5 Examples

Interactively:

<FHELP

This call displays the ALL-IN-1 functions on the terminal.

6.37 FIELD

6.37.1 Syntax

FIELD [field-name[,field-name...]]

6.37.2 Description

Fields on a form are where the actual interchange of data takes place between the user and ALL-IN-1. The FIELD function invokes the Field Processing facility to activate one or more fields on the current form. ALL-IN-1 places the user in the first (or only) field if it is enterable.

You can call FIELD from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.37.3 Function Parameters

field-name...

The optional FIELD parameters are names of one or more enterable fields on the current form. Calling the FIELD function is equivalent to calling the FORM function and specifying the /FIELDS qualifier to activate selected fields. In both cases, you are identifying the fields that participate when ALL-IN-1 displays the form. The order in which you list the fields is irrelevant. ALL-IN-1 processses the fields in the order specified when the form was created.

When you call FIELD without parameters, it activates all participating fields on the form. The effect is the same as if you called the function with the wildcard parameter, "*". In all cases, Field Processing processes the field qualifiers associated in Named Data with a given field when the user touches that field.

6.37.4 Function Qualifiers

None.

6.37.5 Examples

Interactively:

```
<FIELD FNAME, DESC, REPLY
```

is equivalent to

```
<FORM . /FIELDS=FNAME, DESC, REPLY
```

6.37.6 Special Features and Restrictions

Although the FIELD function activates fields on the current form, you can call it when there is no current field-oriented form on the screen. In this case, ALL-IN-1 processes the last active form. The field name must be a valid, enterable (FMS non display-only) field on the form.

In addition, when you call FIELD from the subprocess, the value of the keystroke that the user pressed to complete the field is stored in ALL-IN-1 symbols OA$FORM_TERMINATOR and OA$FORM_DISPOSE.

6.38 FOR

6.38.1 Syntax

FOR RSE DO function[\\function...]

6.38.2 Description

The FOR function searches data sets for records that meet its selection criteria and then executes one or more action functions for each record it finds. You usually call the action functions to display the selected records, but you can call any function or list of functions, including another FOR function.

You can call FOR from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- From within validation expressions

6.38.3 Function Parameters

FOR is a conditional function: it executes the other function or functions specified after the keyword DO for each record that satisfies the record

selection expression (RSE). The RSE is everything between the FOR and the DO keywords. The full FOR syntax is:

```
FOR [UNIQUE field] [FIRST] ds [,ds...] [WITH] exp DO func[\\func...]
```

where

UNIQUE field is an optional keyword and its argument is a field in the data set that FOR checks for uniqueness. Specifying UNIQUE causes FOR to select and act upon only the first record occurrence of a series of records containing identical values in the specified field. The field specified should be the key field of the data set (depending on the key of reference). FOR only looks for duplicates of adjacent records; it does not scan the entire file for duplicates of a value in the specified field.

FIRST is an optional keyword. When you specify FIRST, FOR selects and acts upon only the first matching record. Unlike UNIQUE, FIRST stops searching all data sets when it finds the first match. When you omit FIRST, FOR acts upon all matching records.

ds... are one or more data sets. A data set is usually an entry form that describes an indexed data file. However, the entire File Cabinet, for example, is a data set, and one that is accessed with a special syntax.

WITH is an optional keyword; use it for clarity.

exp is a Boolean expression that you build using a combination of field names, other ALL-IN-1 symbols, Boolean operators, and relational operators. If you omit exp all records are selected.

DO is a required keyword.

func... is a function or list of functions that FOR executes for each record that satisfies the RSE. Since you can specify FOR itself in this function list, use two backslashes to separate each DO function.

6.38.4 Boolean Expressions

That part of the RSE between the WITH and the DO keywords is a Boolean expression, or a set of search instructions that FOR uses to select records from the data set(s).

Create Boolean expressions using the following rules:

.field rel-oper symbol [Bool-oper .field rel-oper symbol ....]

where

.field — is a field in that data set. Usually this is an FMS field on one or more specified entry forms. The dot prefix is required syntax that identifies the field as belonging to the data set (as opposed to the current form).

rel-oper — is a relational operator. Table 6-3 lists the valid relational operators.

Table 6-3 Relational Operators

Numeric	String	Meaning
EQ	EQS, EQUAL, = =	Equal to
LT, <	LTS, BEFORE	Less than
LE	LES	Less or equal
GT, >	GTS, AFTER	Greater than
GE	GES	Greater or equal
NE	NES	Not equal
EN	ENS	Equal or null
	CONTAINING, < = >	Containing
	BEGINNING, = =	Beginning

NOTE: The numeric and string operators compare strings in different ways. Numeric operators compare the numeric value of two strings, while the string operators compare the strings themselves. Thus:

"1234" LT "999" is false,

"1234" LTS "999" is true.

Similarly:

"1" EQ "0001" is true,

"1" EQUAL "0001" is false.

symbol — is an ALL-IN-1 symbol-table symbol, a quoted string, a field on the current form, or a field in the data set.

Bool-oper — is a Boolean operator, either: AND, OR, or NOT.

The minimum Boolean expression is the numeral 1: use it when you want FOR to treat all records in the data set(s) as yes or true matches to the search criteria.

When you specify the .field rel-oper symbol.. type of expression, the relational operators determine whether the expression is evaluated as true or false. Some examples are illustrated in Table 6-4.

Table 6-4 Boolean Expressions

Field	Contains:	Expression				Match?
FIRST	"ABC"	WITH .FIRST	EQS	"ABC"	DO ...	yes
		WITH .FIRST	EQS	"ABCD"	DO ...	no
		WITH .FIRST	CONT	"A"	DO ...	yes
		WITH .FIRST	CONT	""	DO ...	yes*
SECOND	"123"	WITH .SECOND	EQS	"000123"	DO ...	no
		WITH .SECOND	EQ	"000123"	DO ...	yes
		WITH .SECOND	LTS	"11111"	DO ...	no
		WITH .SECOND	LT	"11111"	DO ...	no
		WITH .SECOND	ENS	"123"	DO ...	yes*
		WITH .SECOND	EN	""	DO ...	yes*
THIRD	"PWP11"	WITH .THIRD	EQ	"PWP11"	DO ...	error; invalid numeric

*The expression A CONT B is true if
- B is a substring of A, or
- B is a null string.

*The expression A ENS B is true if
- A exactly matches B, or
- B is a null string.

*The expression A EN B is true if
- A is equal to B,
- B is zero, or
- B is a null string.

You can substitute the alternate form of any relational operator illustrated in Table 6-4.

6.38.5 Special DO Subfunctions

FOR dispatches a special DO subfunction or one or more ALL-IN-1 functions just after it selects each record. Special DO subfunctions are unique to FOR, and you cannot call them as regular ALL-IN-1 functions. They format, display, and allow the user to choose from the selected records. You can alternatively specify any valid ALL-IN-1 function or function list after DO, including another FOR function. Thus, you can nest FOR functions.

When you build a DO list with more than one function, separate each function call with two backslashes. If you use a single backslash, ALL-IN-1 interprets it as the end of the FOR function.

The special DO subfunctions are listed below.

SEL_CHOICE output

SEL_CHOICE displays fields from selected records in the format specified by output and allows the user to select an item from the list. Output is a combination of fields and literal strings.

When FOR calls SEL_CHOICE, this subfunction displays the record collection on the current form in a scrolled region named DISPLAY, for example, on the various Index forms used in ALL-IN-1. This subfunction prompts the user to select one item from the displayed list.

SEL_DISPLAY output

SEL_DISPLAY displays fields from selected records in the format specified by output. Output is a combination of fields and literal strings.

When FOR calls SEL_DISPLAY, this subfunction displays the record collection on the current form in a scrolled region named DISPLAY.

SEL_STYLE output

SEL_STYLE formats fields from selected records as specified by output. Output is a combination of fields and literal strings.

SEL__STYLE only formats; it does not output a list. Use this subfunction in a select form, in conjunction with the /STYLE select-form qualifier. /STYLE determines how the FOR function outputs the SEL__STYLE-formatted records.

The syntax for all the "SEL__" functions' output parameter is identical:

```
expression [expression...]
```

where expression is either:

```
.field[:length[offset]]
```

where the dot is required syntax, length is an optional length that you specify to extract a substring, and, offset is the distance into the string to begin the extract.

OR

```
"string"
```

where string is any literal string, enclosed in quotes. You specify a single blank here as spacing between fields.

You can combine strings and fields: the syntax is identical to that used for the /SHOW field qualifier.

6.38.6 Examples

To display a list of names and titles of a group of ALL-IN-1 users, specify:

```
FOR PROFIL WITH .USER BEGINNING "SMIT" DO SEL__DISP .FULNAM " " .TITLE
```

In this example, ALL-IN-1 first opens the data file associated with entry form PROFIL if the file is not already open. Then, FOR selects all records in this file where field USER begins with 'SMIT'.

SEL__DISPLAY formats the record collection and displays it on the current form in scrolled region DISPLAY.

The dot in .USER is important — it instructs the FOR function to search the USER field associated with form PROFIL rather than the USER field

of the current form. If you omit the dot an error occurs; but you can specify field USER elsewhere in the Boolean expression, and the presence or absence of the dot changes its meaning. For example:

```
FOR PROFIL WITH .USER CONTAINING USER DO SEL__DISP .FULNAM " ".ZIPCOD
```

This selects those PROFILE.DAT records in which USER field of the file contains what is in the USER field in the current form.

A more complex example:

```
FOR PROFIL,MLIST WITH .DEPART = = "PAYROLL" AND .PHONE < = > "(603)"
DO SEL__CHOICES .USER " " .DEPART " " .PHONE
```

This function call searches two data sets: the files associated with entry forms PROFIL and MLIST. All records in either file with the DEPART field equal to "PAYROLL" and the PHONE field containing "(603)" match the selection criteria. In this case, departments specified as "PAYROLL DEPT" do not match. The following phone numbers match:

(603)555-1212,
A/C (603) 555-1212;

and the following do not match:

603-555-1212,
(603-555-1212).

As before, the dots preceding fields .DEPART and .PAYROLL indicate that these are fields on either PROFIL or MLIST.

You can add the keywords UNIQUE or FIRST to further refine the search. Though these keywords are independent of one another, FIRST implies UNIQUE, so specifying both of them causes UNIQUE to be ignored.

For example, you can treat field DEPART as a secondary, nonunique key to file OAUSER:PROFILE.DAT: that is, more than one individual can work in a department. The following FOR function call would retrieve one record — the individual whose username comes first, alphabetically, in either the Personnel or the Payroll departments (assuming that there are two such entries):

```
FOR FIRST PROFIL WITH .DEPART = = "PERSONNEL" OR .DEPART = =
"PAYROLL" DO SEL__DISPLAY .FULNAM " " .PHONE
```

When you search the File Cabinet with FOR, use the CAB$ syntax. The general File Cabinet DSR syntax is similar to that for entry-form-related data sets:

```
FOR CAB$[keyword] WITH .field rel-oper symbol [...] DO function
```

You can also use FOR to search for RMS files on disk. Use the OA$DIR DSR, with the following syntax:

```
FOR OA$DIR:file-spec WITH 1 DO function [\ \function...] ]
```

where file-spec is any valid RMS file specification; wildcard characters are allowed.

You can also use OA$DIR in validation expressions and with the GET function.

6.38.7 Special Features and Restrictions

An RSE may contain the "@" symbol-substitution operator, which performs interactive substitution. When this operator is present, FOR defers its processing until it has selected the record. Thus, you can specify symbol lists within an RSE, and the selection criteria can change dynamically as FOR reads each record. For example:

```
FOR CAB$ WITH ",GENERAL,HOLD," CONT @"',' .FOLDER ','" DO ...
```

This FOR function call finds all folders named GENERAL or HOLD.

FOR frequently requires a default scrolled region called DISPLAY on the current form in which to display records. A scrolled region called DISPLAY is a prerequisite for the select form .TYPE, which calls FOR directly from the .TYPE line of Named Data. The search and select form .TYPES are essentially interfaces for the FOR function.

You can also call FOR indirectly through validation qualifiers. Validation expressions can optionally be expressed as RSEs. Since the validation expression is handled internally as a FOR function call without keywords or special subfunctions, the features covered in this section also apply, in most cases, to /INVALID, /RECOGNITION, and /VALID.

The FOR function can also be used in conjunction with the WRITE function to convert data files when you have changed one or more field names in the file. The general syntax is:

```
FOR old_form DO WRITE ADD new_form field = sym[,field = sym...]
```

where

old_form is the name of the entry form that describes the old data file.

new_form is the name of the entry form that describes the new data file.

field=symbol... are fields on the new form and symbols that contain the values to be placed in those fields. The first field name in the list is always the key to the file.

6.39 FORCE

6.39.1 Syntax

FORCE

6.39.2 Description

The FORCE function causes ALL-IN-1 to display the current form and the current message immediately. Usually ALL-IN-1 defers the display of forms and messages until just prior to accepting input from the user. When you call functions such as PUT_FIELD and DISPLAY to output text to a display-only field or to line 24, that text never appears unless the form has an enterable field that requires user input. FORCE overrides this feature and allows you to use the DISPLAY function, for example, to display the message `Working...` FORCE performs no action, if the current form is on the screen and there are no messages waiting to display.

You can call FORCE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.39.3 Function Parameters

None.

6.39.4 Function Qualifiers

None.

6.39.5 Examples

In Named Data:

Named Data

```
17 Name .COMMA
DISPLAY Press Gold H for HELP\FORCE
```

This example is from the Named Data of form DEFAULT. If you have not redefined keypad COMMA on the current form, then `Press Gold H for HELP` displays when the user presses this key. The FORCE ensures that this message displays under all circumstances.

```
4 Name .GOLD A
OA$FLD_DONE\FORM DOCDS2\IFEXIT\DISP Creating new document...\FORCE
```

This example is from the Named Data of form DOCDSC. FORCE forces the display of a message that the user would otherwise not see.

In a command procedure:

```
$ WRITE OAMAILBOX "OA DISPLAY No file selected\FORCE"
$ @DCLMAILBOX:
```

6.40 FORM

6.40.1 Syntax

FORM [form-name [type][/qual[= arg][/qual[= arg]...]]]

6.40.2 Description

The FORM function invokes the Form Processing facility to retrieve, process, and display the form you give as a parameter. You call forms as they are, or you redefine the form .TYPE by specifying a new type in the function call.

This function can take many qualifiers, some of which are common to all form TYPEs and others of which are specific to one form .TYPE.

The FORM function keeps track of how the user dismisses a form by saving terminator values in up to three symbols: special symbols OA$FRM__DISPOSE and OA$FIELD__TERM, and, in the subprocess, global DCL symbol TERMINATOR. You test these symbols to determine how the user left the form and the exact key sequence used.

You can call FORM from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.40.3 Function Parameters

form-name

You can optionally specify the name of a form to process. When you omit the form-name parameter, the FORM function acts as a displays-previous-form function. You can also specify a substitution character (.) to refer to

the current form. Use the substitution character in a situation like the following:

```
$  !  Call the form as a menu first to get selection
$  !
$  FORM MENARG MENU/CHOICE=CHOICE
   ...
$  !  Now call MENARG as an argument form and let the user steer
$  !  through the fields
$  !
$  FORM . ARG/FIELDS=P,R,N,D,L
```

Form processing interprets the period in the second form call as representing the most recently processed form, which in this case is MENARG. If you had called another form between these two function calls (such as through a /POST form qualifier), then that other form would be the most recently processed form.

The form .TYPE parameter is optional with the substitution character, allowing you to either reprocess a form as the same form .TYPE with a different set of form qualifiers or as a new form .TYPE altogether.

When looking for the form, ALL-IN-1 searches the memory-resident forms first and then all form libraries listed in the User Profile Form Library field (FRMLIB) in the order they are listed. ALL-IN-1 usually searches OA$LIB:OAFORM last; however, you can specify this form library in the User Profile to cause ALL-IN-1 to search it in some other order.

type

You can optionally specify a form .TYPE in the function call. The .TYPE parameter resets the .TYPE directive that the form was created with and allows you to use the form as a different kind of interface.

Calling a form as a new .TYPE causes ALL-IN-1 to ignore all the form qualifiers currently associated with the form. Thus, the /RESET qualifier is

implied. For example, if the Named Data of form EFORM contained the following .TYPE directive:

Named Data

```
1 Name .TYPE
ENTRY/MODE = UPDATE/FIELDS = A,B,C/HIGHLIGHT
```

then the following function call causes ALL-IN-1 to process EFORM with all fields participating:

```
<FORM EFORM ARG
```

The /MODE, /FIELDS, and /HIGHLIGHT, qualifiers are ignored.

6.40.4 Form Qualifiers

As illustrated above, calling a form as a new .TYPE always resets all form qualifiers. When you specify form qualifiers in a FORM function call that does not include a new .TYPE, those qualifiers may either override the currently active qualifiers or they may augment them, depending on the particular qualifier. For example, the /RESET qualifier takes no arguments, but it causes ALL-IN-1 to ignore all currently active qualifiers. /FIELDS, which identifies fields that participate in the form, overrides any previous /FIELDS qualifier; but /GET is appended to the other fields initialized by any previous /GET.

See Chapter 3 for descriptions of form qualifiers and form processing.

6.41 FORMAT

6.41.1 Syntax

FORMAT infile_sym,outfile_sym,device[,settings][,hflag][,aflag]

6.41.2 Description

The FORMAT function formats files for printing or listing. You call FORMAT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts

6.41.3 Function Parameters

FORMAT takes three required parameters and three optional ones:

infile_sym

The first FORMAT parameter is an ALL-IN-1 symbol that represents the name of an input file to ready for output.

outfile_sym

The second parameter is an ALL-IN-1 symbol that represents the name of an output file to be created by FORMAT.

device

The third parameter is an ALL-IN-1 symbol that represents the type of output device on which the file is printed or listed. FORMAT does not send or queue the file to the device; it simply formats it.

The valid device types are represented by the following keywords:

- LA50 — Personal computer printer.
- LA100 — Correspondence-quality printer.
- LP11 — Line printer; this is the default setting.
- LQP02 — Letter-quality printer.
- VT100 — File is formatted for terminal output.

settings

The fourth parameter, which is optional, is a numeric value from −2 to 9 that corresponds to print or display settings. When you omit settings, it defaults to −2.

The settings corresponding to the valid values for this parameter are:

Setting	Description
−2	Do not display print-settings menu; format the document with predefined settings
−1	Display print-settings menu
0 ... 9	Do not display print-settings menu; format the document with setting n

hflag

The fifth FORMAT parameter, which is optional, is the header-line flag and is only meaningful when the input file is a mail message. This parameter may be either 0, which causes FORMAT to format the message's associated header lines, or 1, which causes FORMAT to ignore the header lines. When you omit hflag, it defaults to 0.

aflag

The sixth parameter, which is optional, is the no-attachments flag and is only meaningful when the input file is a compound File-Cabinet document (that is, a document with attachments). This parameter may be either 0, which causes FORMAT to format all attachments, or 1 which causes FORMAT to ignore attachments. When you omit aflag, it defaults to 0.

You may also use FORMAT as the source parameter to either the LIST or MERGE functions:

<LIST <FORMAT parameters

<MERGE <FORMAT parameters

The FORMAT parameters are the same as those described previously, with the exception of the outfile_sym, which must contain the null text string.

See the LIST and MERGE functions in this chapter for more information.

You can also use FORMAT in Pipeline Mode. The syntax is:

<dest function <FORMAT params

6.41.4 Function Qualifiers

None.

6.41.5 Examples

In a command procedure:

```
$  !
$  ! Get the name of the document to format
$  !
$  WRITE OAMAILBOX "OA GET CAB$.FILENAME[$WPPRDOC]"
$  @DCLMAILBOX:
$    file_name := 'result
$  !
$  ! Build a file name for the output document based on
$  ! username and doc-number, give it a ".LIS" extension
$  !
$  WRITE OAMAILBOX "OA GET OA$USER:3 -"
$  WRITE OAMAILBOX "OA CAB$.DOCNUM[$WPPRDOC] "".LIS"""
$  @DCLMAILBOX:
$    list_file := 'RESULT
$  !
$  ! Call the FORMAT function; PTRTYP contains the destination.
$  !
$  WRITE OAMAILBOX "OA FORMAT """'file_name'"",""'list_file'"",-"
$  WRITE OAMAILBOX "OA """'PTRTYP'"",'qmenu',0,0"
$  @DCLMAILBOX:
```

The following command-procedure segment formats a User Defined Process (UDP).

```
$  WRITE OAMAILBOX "OA GET #COPIES"
$  @DCLMAILBOX:
$   COPIES := 'RESULT'
$   file_name = "[.UDP]" + "''P2'" + ".UDP"
$  WRITE OAMAILBOX "OA MAKE_FILE #TEMPFILE,""''p2'"",".LIS"""
$  @DCLMAILBOX:
$  WRITE OAMAILBOX "OA GET #TEMPFILE"
$  @DCLMAILBOX:
$   list_file := 'RESULT
$  WRITE OAMAILBOX "OA FORMAT ""''file_name'"",""''list_file'"",-"
$  WRITE OAMAILBOX "OA ""''PTRTYP'"",-2,0,0"
$  @DCLMAILBOX:
   .....
```

6.41.6 Special Features and Restrictions

FORMAT can not be invoked when control resides in the WPSPLUS editor (For example, when executing a UDP inside the editor). If this is attempted, a file-interlock error results and the FORMAT operation is not done.

6.42 GET

6.42.1 Syntax

GET [dest-symbol = [dest-symbol = ...]]symbol-string

6.42.2 Description

The GET function is a primary tool for retrieving information and assigning values to symbols in ALL-IN-1. You can use GET to:

- Retrieve and display the values of local, permanent, and special symbol-table symbols.
- Retrieve and display the values of fields on the current or most recently processed form.
- Retrieve and display information from a data set with a data set reference (DSR).
- Retrieve and display the value of a document attribute.
- Assign any combination of the preceding symbol types to a symbol-table symbol, an updatable special symbol, or a field on the current form.
- Call a special output destination.

You call GET from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.42.3 Function Parameters

GET parameters are separated by equal signs. GET is unusual in that optional parameters precede a required one. However, this syntax is consistent with how GET retrieves and assigns values to the various types of symbols.

dest-symbol...

When you supply more than one GET parameter, GET treats all but the last one as destination symbols, and it treats the last parameter as the symbol-string.

GET gives all destination symbols the value of the symbol-string. You can specify any local or permanent symbol-table symbol, field on the current form, or special output destination as a destination; you cannot specify system-maintained special symbols, DSRs, and quoted strings.

When you specify one or more permanent symbols as destination symbols, GET creates global DCL symbols for each permanent symbol. GET assigns all the DCL symbols the value of the symbol-string and gives each one the same name (less the $ prefix) as the corresponding permanent symbol.

symbol-string

The symbol-string may be one symbol or a string of symbols which GET concatenates. For example: if Y = "there" and Z = "is", then calling the function GET X = Y ""Z places "there is" into the local symbol X.

GET behaves in the following ways depending on whether you have specified the symbol-string alone or in conjunction with one or more destinations:

- Only one symbol-string is specified — GET determines its value and does one of two things, depending on where you call GET:
  - Called from Named Data, interactively, and so on — GET displays the symbol value on line 24 of the screen.
  - Called from a Command Procedure in subprocess — GET saves the symbol value in global DCL symbol RESULT.

- One or more destinations are specified in conjunction with a symbol-string — GET loads all destination symbols with the value of the symbol-string. Then, GET does one of two things, depending on where you are when you call it:
  - Called from Named Data, interactively, and so on. — No display on line 24.
  - Called from a command procedure in subprocess — If the destination is a permanent symbol, GET creates a corresponding global DCL symbol, as described previously. (The subprocess must be open.) For example:

```
$ WRITE OAMAILBOX "OA GET $AAA = $ZZZ = 'Alpha, Omega'"
$ @DCLMAILBOX:
$ !
$ ! Global DCL symbols AAA, ZZZ are created (or updated)
$ ! and both are given the value "Alpha, Omega"
```

6.42.4 Function Qualifiers

None.

6.42.5 Examples

One parameter:

```
<GET 'A simple line of text'
```

GET repeats the quoted string, less the quotes, on line 24. The result is the same if you enclose the text in double quotes.

```
<GET $IFOLDER
```

GET displays the value of $IFOLDER on line 24. The $ sign identifies $IFOLDER as a permanent symbol-table symbol.

```
$ WRITE OAMAILBOX "OA GET #CHOICE"
$ @DCLMAILBOX:
$ LOCAL_CHOICE := 'RESULT
```

This GET from a command procedure automatically assigns DCL symbol RESULT with the value of local symbol-table symbol #CHOICE. RESULT, a common global symbol, is used by several functions, so the value of #CHOICE is saved in another symbol, LOCAL_CHOICE.

```
<GET OA$TERMINAL:10
```

OA$TERMINAL is a system-maintained special symbol that identifies your terminal type. The OA$ is required syntax, and the :10 is a substring instruction that will cause GET to extract the first 10 characters of OA$TERMINAL. However, since the value of OA$TERMINAL rarely exceeds 5 characters, GET pads the result with blanks.

```
<GET DTR$MODDAT " equals " MODDAT
```

DTR$MODDAT returns the value of field MODDAT in a currently selected DATATRIEVE record. This special symbol is part of a symbol list that concatenates the value of MODDAT to a quoted string and to an FMS field, MODDAT, on the current (or most recently processed) form,

```
<GET FORMAT.DESC["RUNOFF"]
```

This DSR returns the contents of field DESC of the record indexed by the value of "RUNOFF" from the FORMAT data set. This data set consists of entry form FORMAT, which describes data file OA$DATA:FORMAT.DAT. The "." between FORMAT and DESC is required, as are the brackets enclosing the key.

You can also use DSRs to get information about File-Cabinet documents:

```
<GET CAB$:.TITLE[OA$CURDOC]
```

returns the title of the current File-Cabinet document. This is equivalent to:

```
<GET OA$CURDOC_TITLE
```

```
<GET CAB$ATTRIBUTES:TO.VALUE["TO"]
```

returns the first "TO" attribute of the current document from a document attributes file.

DSRs may specify substrings, multiple fields within one record, and alternate keys.

Named Data

```
1 Name FILEPIECE
/POST = 'IFEXIT\GET OA$DIR: "*.*;*".%WHOLE[FILEPIECE]'
```

After the user enters a value in field FILEPIECE on this form, the /POST field qualifier calls GET OA$DIR to return an expanded file specification using the contents of that field as the bracketed file-spec. OA$DIR treats FILEPIECE as a file-name root unless it contains file-spec delimiters like double colons, semicolons, and so on. The file name displays on line 24.

Since single quotes surround the entire /POST function call, double quotes are used in the default file spec.

Multiple parameters:

```
<GET $ABC = "A simple line of text"
```

Permanent symbol $ABC now contains the text string. Nothing displays on line 24.

```
<GET #DEF = "Knee-deep dust"
```

Temporary symbol #DEF now contains the text string. Nothing displays on line 24.

```
<GET $ABC = #DEF
```

Both $ABC and #DEF now contain the string "Knee-deep dust". Global DCL symbol ABC contains the same value.

```
<GET $ABC = #DEF = "Paris in the spring"
```

Both $ABC and #DEF now contain the string Paris in the spring, as does global DCL symbol ABC.

```
<GET $ABC = "Paris in the spring" = #DEF
```

This example is invalid, and GET signals an error. You cannot specify quoted strings as destination symbols.

```
<GET USER = TITLE = $ABC
```

Fields USER and TITLE on the current form now contain the value of $ABC. This GET is equivalent to:

```
<PUT_FIELD USER,$ABC\PUT_FIELD TITLE,$ABC
```

```
<GET USER = #DEF = TITLE
```

Field USER and local symbol #DEF now contain the value of field TITLE on the current form. This GET is equivalent to:

```
<PUT_FIELD USER,TITLE\GET #DEF=TITLE
```

```
$  ! Create a folder name from the contents of DCL symbol FNAME.
$  !
$  FNAME = "ANYTHING.TXT"
$  !
$  FOLDER := "Folder "F$EXTRACT(0,F$LOCATE(".",FNAME),FNAME)'"
$  !
$  ! FOLDER = "Folder ANYTHING"; Use GET to assign a value
$  ! to ALL-IN-1 symbol $WPFDR from the contents of FOLDER and
$  ! of ALL-IN-1 symbol #SEQUENCE.
$  !
$  WRITE OAMAILBOX "OA GET $WPFDR=""'FOLDER'"" #SEQUENCE"
$  @DCLMAILBOX:
$  !
$  ! Assume #SEQUENCE contains "123"; $WPFDR now equals
$  ! "Folder ANYTHING123". Global DCL symbol WPFDR is assigned
$  ! the same value.
$  !
$  FOLDER := "'WPFDR'"
$  !
$  ! DCL symbol FOLDER now equals "Folder ANYTHING123".
$  ! DCL symbol RESULT, not WPFDR, is given the value of $WPFDR
$  ! in the following GET,
$  !
$  WRITE OAMAILBOX "OA GET $WPFDR"
$  @DCLMAILBOX:
$  FOLDER := "'RESULT'"
```

The preceding command procedure illustrates how GET assigns values to DCL symbols from the subprocess.

```
<GET OA$DCL="$SET HOST " PROFILE.NODE[#OTHER_USER]
```

OA$DCL builds a command line with the symbol list, and then sends that line to the subprocess as a SET HOST command.

```
<GET OA$DISPLAY = FIELDC " can't be found in " $IN_FILE
```

OA$DISPLAY builds a DISPLAY function call with the symbol list and then displays the translated expression on line 24.

```
<GET OA$FUNCTION = "FORM " $NEXT_FORM_TO_DISPLAY
```

OA$FUNCTION builds a function call with the symbol list and then calls the FORM function with the contents of permanent symbol $NEXT_FORM_TO_DISPLAY.

```
<GET OA$FUNCTION = "DISPLAY " FIELDC " can't be found in " $INFILE
```

This example is functionally identical to the preceding OA$DIR example.

```
<GET OA$PROMPT_HELP = "GENERAL RECOG"
```

GET assigns special symbol OA$PROMPT_HELP the value "GENERAL RECOG", which Help then uses as a keyword topic-list the next time you call the HELP function from a prompt.

```
<GET #FILNAM = OA$DIR:"*.*;*".%WHOLE["MYFILE"]
```

GET gives #FILNAM the value returned by OA$DIR, which searches the default directory for a file with a file-name root of "MYFILE".

6.42.6 Special Features and Restrictions

When you GET a data set reference from the subprocess that has multiple fields, the destination DCL symbol (either RESULT or some other symbol) will contain a string consisting of the concatenated field values. For example:

```
$  ! Get the first and second address lines from the user profile entry
$  ! for user NAME. Store these lines in symbol FULL_ADDRESS.
$  !
$  WRITE OAMAILBOX "OA GET PROFIL.[""'NAME'""] .ADDR1 "", "" .ADDR2"
$  @DCLMAILBOX:
$  FULL_ADDRESS :== "'RESULT'"
```

6.43 HELP

6.43.1 Syntax

HELP [keyword_list_symbol]

6.43.2 Description

The HELP function opens a help library and displays help text based either on specified keywords or on the current ALL-IN-1 context. ALL-IN-1 Help uses VMS Help libraries, which are organized hierarchically into text modules. These modules are numbered and indexed by keyword, and these keywords are the optional parameters of the HELP function.

HELP displays text through a scrolled region on special memory-resident help forms.

You call HELP from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.43.3 Function Parameters

keyword_list_symbol

You may optionally specify a topic path containing up to eight topics in the form of an ALL-IN-1 symbol or symbol list. The contents of keyword_list_symbol takes the form:

level-1-topic [level-2-topic ... level-8-topic]

The top-level topic corresponds to a level 1 module of help text. When you omit this parameter, it defaults to different values, depending on the context in which you call help.

ALL-IN-1 form names correspond to keywords in a modified VMS help library, OA$LIB:OAHELP.HLB. The help topic path is a tree: each top-level topic can have one or more level-2 subtopics, each of which in turn have level-3 subtopics, and so on. For example, the following interactive call:

```
<HELP 'WP C'
```

displays Help on the Create option (level-2 topic) of form WP (level-1 topic). You can only use double quotes to enclose a literal string when calling HELP.

When you call HELP, it searches the topic tree, and if it finds the associated text, it displays it. If it does not find the text, it trims the last keyword off the topic path and searches the library again with the shortened keyword list. If this continues until there are no keywords left, HELP displays a list of additional topics. Otherwise, HELP displays the additional-topics list after the user finishes reading the help text. Control returns to the point of call only when the user exits Help with EXIT SCREEN.

Help displays in a scrolled region on the form:

OA$HELP_TOP — A half-screen overlay that displays when the cursor is on the bottom half of the screen when you call help.

OR

OA$HELP_BOTTOM — A half-screen overlay that displays when the cursor is on the top half of the screen when you call help.

An entry in the Named Data of form DEFAULT associates Gold H, the HELP key, with a HELP function call with no parameters.

6.43.4 Function Qualifiers

None.

6.43.5 Examples

Interactively:

```
<HELP WP
```

Calling HELP interactively from a menu choice field is equivalent to pressing Gold H in that field. Help displays on form WP.

In Named Data:

Named Data

```
1 Name .TYPE
VIEW MESSAGES /PRE = 'GET #PTEMP = OA$PROMPT_HELP

2 Name .TYPE
\GET OA$PROMPT_HELP = "ERRORS " OA$MSGID'

3 Name .TYPE
/POST = 'OA$PROMPT_HELP = #PTEMP'
```

This sequence does not illustrate a HELP function call, but it does illustrate how HELP is provided from form OA$VIEW_MESSAGES, which displays when you press Gold W. There are no enterable fields on OA$VIEW_MESSAGES, but there is a prompt at which the user may press Gold H. When the user calls HELP from any prompt, HELP looks first for keywords in special symbol OA$PROMPT_HELP. The preprocessing GETs done in OA$VIEW_MESSAGES load this symbol with the keyword that displays HELP on the last error message if the user presses Gold H at the next prompt.

6.43.6 Special Features and Restrictions

You can specify a string with no quotes as a HELP parameter, and HELP treats it as a literal if it cannot translate it as a symbol.

HELP looks for main-process logical OAHELP before it opens the default help library, OA$LIB:OAHELP.HLB. If OAHELP is assigned, HELP translates it and treats its equivalence name as a VMS Help library name.

As illustrated in the preceding section, you can assign special symbol OA$PROMPT_HELP a value that HELP uses as a keyword list if the user calls HELP from a prompt. If the user calls HELP from a prompt and OA$PROMPT_HELP is null, HELP looks for the topic "current-form-name PROMPT".

6.44 IFEXIT

6.44.1 Syntax

IFEXIT

6.44.2 Description

The IFEXIT function is a switch, to use with stacked functions, to control the flow of processing. When present in a function list, IFEXIT causes ALL-IN-1 to ignore the functions that follow it if the user presses EXIT SCREEN while responding to a function that precedes IFEXIT.

IFEXIT is only meaningful when functions are stacked on either side of it, and only then when a function that precedes it can be terminated with an EXIT SCREEN, a Gold Q, or some other key sequence that you have associated with OA$FLD_EXIT. The functions that can be dismissed with a call to OA$FLD_EXIT are:

- EDIT
- FORM
- FIELD
- PROMPT
- YESNO_PROMPT

6.44.3 Function Parameters

None.

6.44.4 Function Qualifiers

None.

6.44.5 Examples

Interactively:

```
<FORM EMC \IFEXIT \ FORM EMGETD
```

IFEXIT checks whether the user left form MAILARG through the EXIT SCREEN; ALL-IN-1 continues or ends processing based on user response.

6.44.6 Special Features and Restrictions

An entry in the Named Data of form DEFAULT associates keypad 0 with a call to the OA$FLD_EXIT function. IFEXIT actually looks for the disposition that OA$FLD_EXIT sets, not the key that the user presses to call OA$FLD_EXIT. Thus, if you associate keypad 0, for example, with OA$FLD_DONE locally, then ALL-IN-1 sets the DONE disposition, not EXIT, when the user presses EXIT SCREEN.

6.45 INCLUDE

6.45.1 Syntax

INCLUDE form, filename__field

6.45.2 Description

INCLUDE includes a File Cabinet document into the editor or into another ASCII file that an application is processing in the subprocess.

6.45.3 Function Parameters

form

The required form parameter is the name of the form that prompts the user for the name of the File Cabinet document to include. This should be an argument form (or be processed as one).

filename__field

The required filename__field parameter is the name of the field in the form which contains the RMS filename of the document to include.

6.45.4 Function Qualifiers

None.

6.45.5 Examples

In an editor initialization file:

The default ALL-IN-1 editor is initialized with a file that calls the INCLUDE function as follows:

```
DEFINE MACRO GET
INSERT=GET
INCLUDE "CHANGE;XLATE OA INCLUDE EDTINC,FILENAME\CLOSE_PRIOR^Z;
REFEXFIND ."
^Z
FIND=MAIN.
^Z
DEFINE KEY GOLD G AS "EXT GET."
```

Pressing Gold G in the editor invokes macro GET, which calls the INCLUDE function to display form EDTINC. The Named Data of form EDTINC is similar to that of form DOCSEL, with the addition of a enterable field, FILENAME, for the RMS file name of a document. This INCLUDE call brings the file into the main edit buffer.

6.45.6 Special Features and Restrictions

When you call INCLUDE from an editor in the main process, ALL-IN-1 includes the document into main edit buffer; in all other cases, INCLUDE sends the document through DCLMAILBOX, under the assumption that you are calling INCLUDE through OAMAILBOX from an image in the subprocess.

The MERGE-template &INCLUDE directive provides a similar capability for MERGE-created files.

6.46 KEYPAD

6.46.1 Syntax

KEYPAD { ON | OFF }

6.46.2 Description

Call the KEYPAD function with one of two keywords to activate or deactivate a terminal's keypad. This function may not be used to control input on a form. For information on using the KEYPAD function with the /KEYPAD qualifier, see the /KEYPAD form qualifier in Chapter 3.

You call this function from the following locations:

- In Named Data
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.46.3 Function Parameters

ON | OFF

The single, required parameter is a keyword, either ON or OFF. KEYPAD ON enables, or activates, the keypad so that ALL-IN-1 interprets its keys as described throughout this APR; KEYPAD OFF deactivates the keypad.

For example, if you press keypad 0 while the keypad is activated, ALL-IN-1 interprets this key as EXIT SCREEN because a .KEY directive in the Named Data of form DEFAULT defines keypad 0 as such. If you press the same key while the keypad is deactivated, your terminal echoes a 0.

6.46.4 Function Qualifiers

None.

6.46.5 Examples

In Named Data:

Named Data

```
10 Name .GOLD $
KEYPAD OFF\COMMAND CALCAPPL\KEYPAD ON
```

This keypad definition associates Gold $ with the three function calls. The COMMAND function call that runs CALCAPPL.COM is bracketed by two keypad calls: KEYPAD OFF to deactivate the keypad before the application runs, and KEYPAD ON to reactivate the keypad after the user returns to the main process.

6.46.6 Special Features and Restrictions

You can use KEYPAD ON to reactivate a keypad that has has been switched off by an application running in the subprocess. You may also need to call this function during or after file transfer or communications between ALL-IN-1 and a personal computer used as an ALL-IN-1 (or VAX) terminal.

The /KEYPAD form qualifier overrides the KEYPAD function.

6.47 LIGHTS

6.47.1 Syntax

```
LIGHTS number[,number...]
```

6.47.2 Description

The LIGHTS function turns the terminal indicator lights on or off.

You can call LIGHTS from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.47.3 Function Parameters

number

The single parameter is a number from zero to four. A zero turns off all LEDs. The number 1 turns on LED L1, the number 2 turns on LED L2 and so on.

6.47.4 Examples

In Named Data

Named Data

```
1 Name LEDON
GET OA$FUNCTION = "LIGHTS " #LED
```

6.47.5 Special Features and Restrictions

This function is only meaningful with VT100 terminals.

6.48 LIST

6.48.1 Syntax

LIST file-name

6.48.2 Description

The LIST function prints a displayable file or pipeline-source output, 23 lines at a time, either on the terminal or an attached printer port. LIST is similar to the DCL TYPE command, but LIST gives the user more control over the display.

You call LIST from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

You can also call LIST in Pipeline Mode to display the output of a pipeline source function as it is generated. The syntax is:

LIST <source_function

6.48.3 Function Parameters

file-name | <source_function

The single, required LIST parameter takes one of these formats:

- The name of the RMS file to list or print. You can specify a device and directory, and the file must be in a displayable (ASCII sequential or graphics) format. You cannot LIST binary files such as object modules and executable images.

The following file types are supported:

- ASCII sequential text files
- VT100 graphics
- REGIS graphics
- SIXEL graphics

- A pipeline source function, preceded by an angle bracket. The valid source functions are MERGE and MERGE_LINE. You can use the pipeline mechanism to bypass intermediate file creation and function calls.

NOTE: The output parameter of MERGE is null. This causes the output to be directed through the pipeline.

If the terminal has a printer port and you have activated the port with the PORT_ON function, then LIST outputs only to the attached printer.

6.48.4 Function Qualifiers

None.

6.48.5 Examples

In Named Data:

Named Data

```
1 Name PS
PORT_ON\LIST TEMP.TXT\PORT_OFF
```

Menu option PS first turns on the printer port so that LIST sends file TEMP.TXT to the attached printer. After the file has been listed, PORT_OFF is called.

Interactively:

```
<LIST OA$LIB:OAINI.COM
```

This LIST function call clears the screen and then lists command procedure OA$LIB:OAINI.COM.

In a command procedure:

```
$  ! List the file associated with DCL symbol PRINT_FILE
$  !
$  WRITE OAMAILBOX "OA LIST "PRINT_FILE""
$  @DCLMAILBOX:
```

In Pipeline Mode:

```
<LIST <MERGE COMPILE.BLP,,NAMES.TMP
```

As MERGE processes COMPILE.BLP and NAMES.TMP, LIST displays the generated output.

6.48.6 Special Features and Restrictions

Although LIST does not recognize ALL-IN-1 symbols as parameters, you can use GET OA$FUNCTION to build a LIST function call with symbols. For example:

```
<GET OA$FUNCTION = "LIST " $LIST_FILE
```

At the end of every screen during a terminal listing, the function prompts the user to press RETURN to continue or EXIT SCREEN to exit.

LIST recognizes the following keys which are defined on the OA$LIST form. They provide special functionality when in the List function:

Key	Function Called	Action
Gold C	OA$LIST_NOPAUSE	Continuous list - no prompts
CTRL/C		Stops continuous list output and returns to normal list prompts
Up	OA$LIST_PAGE_TOP	Go to the top of the page
Gold Up	OA$LIST_TOP	Go to the top of the file
Gold Left	OA$LIST_LEFT	Scroll text left 8 characters
Gold Right	OA$LIST_RIGHT	Scroll text right 8 characters
Down	OA$LIST_PAGE_FORWARD	Go to the next page
Gold E	OA$LIST_WIDTH	Toggle between 80 and 132 character srceen sizes
Gold W	OA$LIST_REFRESH	Refresh the screen

If you list the file to the end, the message:

```
n lines printed -- Press RETURN
```

displays, where n is the total number of lines in the file.

6.49 LOG

6.49.1 Syntax

LOG text

6.49.2 Description

This function adds entries to the ALL-IN-1 Logging facility. The entry is date- and time-stamped before it is sent to the OA$LOGGER MAILBOX. The OA$LOG_TO_FILE function takes the contents from the MAILBOX and writes it to a file specified by filename.DAT.

You call LOG from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In scripts

6.49.3 Function Parameters

text

The single, required LOG parameter is any character string you want to enter in the log file.

6.49.4 Function Qualifiers

None.

6.49.5 Examples

In Named Data:

Named Data

```
1 Name DI
GET OA$FUNCTION = "LIST " LISTFILE \ LOG "Option DI on form ACCT"
```

Option DI lists the file named in field LISTFILE and then logs a message in the log file.

Interactively:

```
<LOG "Form NEWMEN displayed normally"
```

6.49.6 Special Features and Restrictions

Although LOG does not recognize ALL-IN-1 symbols as parameters, you can use GET OA$FUNCTION to build a LOG function call with symbols. For example:

```
<GET OA$FUNCTION = "LOG Logging an entry for user " OA$USER
```

The Logging facility must be running and the Logging mailbox created if you want to write log entries to disc. Both of these conditions must be met for the function to work.

6.50 LOGICAL

6.50.1 Syntax

LOGICAL name[,equiv-name]

6.50.2 Description

Use LOGICAL to define, change, or inquire about logical names in the ALL-IN-1 main process. This function does not affect logicals defined in the subprocess.

You call LOGICAL from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.50.3 Function Parameters

The specific task that LOGICAL performs depends on whether you supply one or two arguments.

name

You must specify at least a logical name, minus any colon that may be part of that name. When you specify only the name, LOGICAL returns the equivalent name. The equivalent name is sent to the current output destination (normally line 24 of the current form). LOGICAL only stores it in the DCL symbol RESULT when you call the function from the subprocess.

equiv-name

To define or redefine a logical, specify the logical's equivalent name as the second, optional parameter. When you supply this parameter, LOGICAL does not output anything and does not alter the DCL symbol RESULT.

6.50.4 Function Qualifiers

None.

6.50.5 Examples

In Named Data:

Named Data

```
1 Name SHL
LOGICAL OA$LIB
```

Menu option SHL displays the equivalent name of logical OA$LIB to the current output destination.

Interactively:

```
<LOGICAL OAUSER
```

Your ALL-IN-1 default directory displays.

```
<LOGICAL OA$LIB,DISK$:[OA.DEVELOPMENT]
```

LOGICAL reassigns OA$LIB to [OA.DEVELOPMENT] subdirectory on device DISK$.

```
<LOGICAL NEWUSER,EDWARDS
```

LOGICAL creates logical name NEWUSER and assigns it the equivalent name EDWARDS. The assignment is stored in the process logical name table and remains valid until ALL-IN-1 exits.

In a command procedure:

```
$ ! Get the user's default directory, store it in symbol DEFAULT.
$ !
$ WRITE OAMAILBOX "OA LOGICAL OAUSER"
$ @DCLMAILBOX:
$ DEFAULT := = 'RESULT
$ !
$ ! Create a logical that defines an alternate File-Cabinet
$ ! index file.
$ ! Ensure that the file is in the user's default directory.
$ !
$ WRITE OAMAILBOX "OA LOGICAL NEW_FILE_CAB,''DEFAULT':NEWDOC.DAT"
$ @DCLMAILBOX:
$ !
```

6.50.6 Special Features and Restrictions

Although LOGICAL does not recognize ALL-IN-1 symbols as parameters, you can use GET OA$FUNCTION to build a LOGICAL function call with symbols. For example:

```
<GET OA$FUNCTION = "LOGICAL " $CURDEF
```

returns the equivalent name of the logical stored as the value of symbol $CURDEF.

```
<GET OA$FUNCTION = "LOGICAL " $CURDEF "," $NEWDEF
```

assigns the value of symbol $NEWDEF to the logical name stored as the value of symbol $CURDEF.

NOTE: *ALL-IN-1 creates logical names OALIB, OADATA, and OAUSER both in the subprocess and in the main process during initialization. If you assign values to OA$LIB and OA$DATA before ALL-IN-1 initializes, it uses the existing values to create the process and subprocess logicals. Thus, you can translate and change these two ALL-IN-1 defined logicals with LOGICAL; you cannot use this function to access or modify other logicals which ALL-IN-1 has defined for the subprocess.*

Calling LOGICAL with an existing logical name as the first parameter and a null string as second parameter does not delete a logical: it simply assigns it a null value. Once LOGICAL assigns a value, the logical cannot be deleted until ALL-IN-1 exits (and then it is deleted automatically).

6.51 MAIL

6.51.1 Syntax

MAIL subfunction

6.51.2 Description

You call MAIL from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.51.3 Subfunction Descriptions

The MAIL subfunctions are described in Chapter 11.

6.52 MAKE__FILE__NAME

6.52.1 Syntax

MAKE__FILE__NAME [/qual] result__sym[,[basis__sym][,def__sym]]

6.52.2 Description

MAKE__FILE__NAME creates a unique RMS file name using information supplied as parameters.

You can call MAKE__FILE__NAME from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.52.3 Function Parameters

result__sym

The first (and only required) MAKE__FILE__NAME parameter is an ALL-IN-1 symbol-table symbol or a field on the current form that receives the created file name. If result-sym is a permanent symbol, ALL-IN-1 creates (or updates) a global DCL symbol and gives it the same value and name (less the "$" prefix) as result__sym.

basis__sym

The second MAKE__FILE__NAME parameter is any ALL-IN-1 symbol containing the basis of the new file-name root. ALL-IN-1 concatenates the first 30 nonspecial characters and uses the result as the file root. Characters accepted in basis__sym are alphanumerics, $(dollar) and __(underscore). Spaces are also accepted, but are converted to underscore.

When you omit basis_sym or if the stripped file-root combined with the contents of def_sym duplicates an existing file name, then ALL-IN-1 creates a unique file-name root based on the current date and time. MAKE_FILE_NAME derives the 9-character alphabetic sequences that serve as default file-name roots from the current date and time.

def_sym

The third MAKE_FILE_NAME parameter is an ALL-IN-1 symbol containing default values to use in creating the file name. The def_sym parameter may be any full or partial RMS file specification. You can insert a dummy file-root for clarity but MAKE_FILE_NAME ignores it since the function itself is creating this part of the name. A full file specification thus has the syntax:

node::device:[directory]optional-root.extension

If you omit def_sym, MAKE_FILE_NAME creates the file name without a file-type extension. Also, since there is no additional default information, any file subsequently created using this name resides in the current default directly (usually OAUSER).

Special symbol OA$DEFAULT_EXTENSION is commonly used as a def_sym parameter. This symbol is a file-type extension derived from the EDITOR field in the User Profile. The recognized editor styles and their associated extensions are listed in Table 6-5.

Table 6-5 Recognized Editor Styles

Style	Internal_Structure	RMS File Extension (OA$DEFAULT_EXTENSION)
DECTEXT	DECTEXT	.DOC
DECTYPE	DECTYPE	.DTP
DPE/EDT	ASCII	.TXT
DPE/WPS	ASCII	.TXT
WPS-PLUS	WPS-PLUS	.WPL
WPS-8	DX	.DX8
other	ASCII	.TXT

NOTE: The DPE|EDT and DPE|WPS editor styles are the two variations of the default ALL-IN-1 editor. All other editors are optional and may not be available on your system.

6.52.4 Function Qualifiers

/NOCHECK

The single, optional qualifier to MAKE__FILE__NAME is /NOCHECK. If you supply this qualifier, MAKE__FILE__NAME puts the created file name into result-sym without checking that the created file name is unique.

6.52.5 Examples

Interactively:

```
<MAKE__FILE__NAME $EDITFILE,"$URELY__YOU JE$T",".TXT"
```

This interactive call assigns $EDITFILE the value "$URELY__YOU__JE$T.TXT", unless the file name $URELY__YOU__JE$T.TXT exists in the user's file directories. If the file name already exists, MAKE__FILE__NAME instead assigns a new, unique file name. MAKE__FILE__NAME also assigns this value to global DCL symbol EDITFILE.

```
<MAKE__FILE__NAME/NOCHECK $EDITFILE,"$URELY__YOU JE$T",".TXT"
```

This interactive call assigns $EDITFILE the value "$URELY__YOU__JE$T.TXT", even if the file name $URELY__YOU__JE$T.TXT exists.

In Named Data:

Named Data

```
1 FILNAM
/POST='MAKE__FILE__NAME FILNAM,USER,OA$DEFAULT__EXTENSION'
```

After the user exits field FILNAM, the /POST field qualifier calls MAKE__FILE__NAME to build a file name using the contents of field USER as a file-root and the contents of OA$DEFAULT__EXTENSION as the file-type extension. MAKE__FILE__NAME stores the resulting file name in field FILNAM.

The following example does the same thing; the only difference is when the file name is created:

```
1 FILNAM
/DOC=USER,OA$DEFAULT__EXTENSION
```

In a command procedure:

```
$!
$! Get the name of the format assoc. with current document
$!
$ WRITE OAMAILBOX "OA GET CAB$.FORMAT[OA$CURDOC]"
$ @DCLMAILBOX:
$!
$! Create a file name using the returned value
$!
$ WRITE OAMAILBOX "OA MAKE__F $NEWNAME,""''RESULT'"",".EDT"""
$ @DCLMAILBOX:
$!
$! Global symbol NEWNAME now contains the newly built file name.
$! Call the editor to create the new file
$!
$ WRITE OAMAILBOX "OA EDIT """'NEWNAME'"""
$ @DCLMAILBOX:
```

6.52.6 Special Features and Restrictions

The /DOC qualifier uses the generalized MAKE__FILE__NAME routines and thus follows the same syntax rules and has the same defaults as MAKE__FILE__NAME.

6.53 MERGE

6.53.1 Syntax

MERGE template_file [,outfile] [,listfile] [,hfile]

NOTE: The outfile parameter is not always optional. See the description of the parameter for the circumstances where it is optional.

6.53.2 Description

MERGE is a list-processing function that creates an output file by processing directives in a template. You can optionally specify a sequential list file with the MERGE function and access this file in the template through special ALL-IN-1 symbols.

Also, you can use the related MERGE_LINE function in conjunction with MERGE to help build the output file.

You call MERGE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts

You can also use MERGE as a pipeline source function. In Pipeline Mode, ALL-IN-1 feeds the MERGE-generated output directly to another function, typically for screen or printer output.

6.53.3 Function Parameters

The MERGE function has one required parameter and three optional parameters.

template_file

The first, required parameter in the function call is the RMS file name of a template, or prototype of the document that you want to create. The template may include both background text and directives. MERGE outputs the background text exactly as it appears in the template. Use the directives to:

- Process variable data, including the optional list file
- Call other ALL-IN-1 functions during the MERGE
- Format the output document

By convention, templates have the file-type extension ".BLP".

outfile

The second parameter in the function call, which is optional, is the RMS file name of the output file. The outfile parameter is optional only when you call MERGE under the following circumstances:

- From within another template (that is, while another MERGE is already active). When no output file is specified, the MERGE output becomes part of the template from which the MERGE has been called.
- From within the editor. When no output file is specified here, MERGE places the output in the current text buffer.
- As a pipeline source function. In this case, the outfile parameter is not allowed. The MERGE output goes directly to the function you have called in Pipeline Mode.
- When you have optionally directed MERGE output to the terminal by calling OA$MRG_TTOUT before MERGE. In this case, if you omit the outfile parameter, MERGE directs its output to the terminal. If you provide the parameter, MERGE ignores OA$MRG_TTOUT and sends its output to the specified file.

  MERGE behaves like this until you call OA$MRG_TTOUT OFF. If you then call MERGE interactively or from Named Data with no outfile parameter, nothing occurs.

When you have specified an output file, it serves as the final product of the MERGE, containing both the background text as it appears in the template and the results of all template directives.

listfile

The third MERGE parameter, which is optional, is the name of a sequential RMS file to merge with the template. MERGE reads the list file line by line, processes the template for each line in the list, and then appends the results to the current contents of the output file. Wherever you have specified OA$MERGE_KEY in a template directive, MERGE replaces it with the current list-file line.

You need not reference OA$MERGE_KEY anywhere. If you omit it and still provide the list file parameter, then MERGE simply uses the list file as a counting mechanism during the merge. That is, MERGE processes the template once for each line in the list file.

Although MERGE recognizes special symbol OA$MERGE_REC in the same way as OA$MERGE_KEY, it is only supported for compatibility with older versions of ALL-IN-1. Use OA$MERGE_KEY when creating new templates.

hfile

The fourth MERGE parameter, which is optional, is the RMS file name of a header file. MERGE merges this header file into the output file whenever the function generates, or encounters a user-specified page break.

MERGE treats the header file exactly like a template. This file may contain both background text and directives, and up to the number of lines that are specified by OA$PAGE_LENGTH (a special symbol that identifies the page size of a MERGE output file; this page size is set to 57 when ALL-IN-1 first runs.)

MERGE merges the header file into the output file at the following points in the merge process:

- At the beginning of the output file, unless the header file begins with the <&NO_PAGE_1> directive
- When you have provided an explicit page break in the template using a <&PAGE> directive
- If OA$PAGE_LENGTH is greater than zero, and if "OA$PAGE_LENGTH" lines have been merged into the output file since the last page-break, or since the beginning of the output

You can change OA$PAGE_LENGTH as desired. MERGE maintains another special symbol, OA$PAGE_NUMBER. This symbol has a value of 1 when you first call MERGE, and MERGE increments OA$PAGE_NUMBER by 1 each time it performs a page-break.

6.53.4 Template Directives

Template directives are processing instructions that MERGE executes or dispatches. As MERGE reads directives from the template, it replaces them in the output file with the output, if any, that the directive generates.

Enclose template directives in <angle brackets>. Directives have one of three formats:

<symbol>

OR

<&OA function[\function...]>

OR

<&special-directive>

The following paragraphs describe these formats.

<symbol>

You enclose any ALL-IN-1 symbol in angle brackets. MERGE translates the symbol and substitutes its value in the output file. For example:

ALL-IN-1 username:	<OA$USER>
Full name from the user profile for the username that is the current value of OA$MERGE_KEY:	<PROFIL.FULNAM[OA$MERGE_KEY]>
Field BBB from the current form:	<BBB>

<&OA function[\function...]>

You can call one or more ALL-IN-1 functions from within a template during a MERGE, including another MERGE. The "&OA" is a required prefix.

<&special-directive>

Special MERGE directives are identified by the prefix "&":

<&INCLUDE file-sym>

The INCLUDE directive includes the text of the specified file into the output file. The file-sym parameter may be any ALL-IN-1 symbol. For example:

Include a temporary file:	<&INCLUDE "TEMP.TMP">
Include the file that is identified by OA$MERGE_KEY:	<&INCLUDE OA$MERGE_KEY>
Include a File-Cabinet document:	<&INCLUDE DOCDB.FILNAM[#DOC]>

<&NO_PAGE_1>

Use the NO_PAGE_1 directive in header files to suppress header-file processing at the beginning of the output file. However, if you begin the main template with "<FF>" or "<&PAGE>", you override the NO_PAGE_1 directive.

This directive is only meaningful in header files.

<&PAGE>

Use the PAGE directive to insert a form feed in the output file.

<&TAB number>

Use the TAB directive to format the output file. The number parameter is the number of spaces to insert at the current position. For example:

<OA$MERGE_KEY:9:0><&TAB 9><OA$MERGE_KEY:30:39> (<OA$MERGE_KEY:30:9>)

<&CLEAR>
<&BOLD>
<&REVERSE>
<&UNDERLINE>
<&BLINK>

Use these directives to control the video attributes of the text to merge. CLEAR clears all other attributes.

You can continue directives over several lines by terminating them with "<->". For example:

```
<&OA DTR SET DICTIONARY CDD$TOP; READY ACCOUNTS AS ACC SHARED WRITE;><->
<&OA DTR FIND ACC WITH NAME="ALLIED" AND ADDRESS CONT "NH"; SELECT>
```

In this example, one DTR function call executes three DATATRIEVE commands.

If a directive is invalid, MERGE ignores it and outputs a single blank.

6.53.5 Function Qualifiers

None.

6.53.6 Examples

In Named Data:

Named Data

```
1 Name B
MERGE OA$LIB:AIBRIEF.BLP,AIBRIEF.LIS,ACTITEM.SEL\QUEUE_PRINT
2 Name B
 "AIBRIEF.LIS"
3 Name F
GET OA$FUNCTION="MERGE PH" $PH_SHORT_DIR "FULL.BLP,
4 Name F
PH" $PH_SHORT_DIR "FULL.LIS, PH" $PH_SHORT_DIR "LIST.SEL"
5 Name F
\GET OA$FUNCTION="QUEUE_PRINT PH" $PH_SHORT_DIR "FULL.LIS"
```

Menu option B processes template OA$LIB:AIBRIEF.BLP, using ACTITEM.SEL as the input list file and creating output file AIBRIEF.LIS.

Menu option F calls the GET function, which in turn calls MERGE indirectly through the special destination OA$FUNCTION. Option F calls MERGE with parameters that have the following general syntax:

```
MERGE PHxxxFULL.BLP, PHxxxFULL.LIS, PHxxxLIST.SEL
```

where xxx is the value of permanent symbol $PH_SHORT_DIR.

Interactively:

```
<MERGE BOILERPLT.BLP,CHAPTER.TXT,TEMP.TMP
```

This interactive function processes the directives in BOILERPLT.BLP and merges it with file TEMP.TMP. MERGE outputs the results to file CHAPTER.TXT.

In a template:

```
<&OA MERGE INPUT.TXT,,TEMP.TMP>
```

This directive merges template INPUT.TXT with list file TEMP.TMP, and MERGE includes the results into the current template.

In an editor initialization file:

```
INCLUDE "OA MERGE OA$LIB:MAILBP.TXT"
```

This callable variation of the editor INCLUDE command calls the MERGE function with one argument: the template. Since you would specify this command while editing, MERGE directs the output into the current edit buffer.

As a pipeline source function:

```
<LIST <MERGE "OA$LIB:REPORT.BLP"
```

This LIST function call displays MERGE output as it is generated.

6.53.7 Special Features and Restrictions

Although MERGE does not translate symbols given as parameters, you can call the function through GET OA$FUNCTION to build a MERGE call from a list of symbols.

If you want to modify a template file (also called a boilerplate) which is contained in a TXL, you must follow some special steps. See Chapter 15 for information on modifying TXL elements.

6.54 MERGE_LINE

6.54.1 Syntax

MERGE_LINE line

6.54.2 Description

MERGE_LINE writes an output line either to a file that has been opened by the MERGE function or to the terminal (if you have called OA$MRG_TTOUT). You can call MERGE_LINE from the same locations that you can call MERGE; however, how and when you call the function is more important than where you call it from.

Call MERGE_LINE in the following situations:

- While the MERGE function is building an output file
- From within the editor to write a line to the file you are editing
- Interactively, after calling OA$MRG_TTOUT, to test directives
- As a pipeline source function to generate a line in Pipeline Mode

6.54.3 Function Parameters

line

The single, required MERGE_LINE parameter is a line of text to include in the MERGE output file. If the line contains angle brackets (< >), MERGE treats the text between those angle brackets as a directive and processes it. If the directive is invalid, MERGE_LINE outputs a single blank.

6.54.4 Function Qualifiers

None.

6.54.5 Examples

In Named Data:

Named Data

```
2 Name .TYPE
ARG/PRE = 'MERGE_LINE Name: '

3 Name NAME
/POST = 'GET OA$MERGE = NAME '
```

This ARG form, called during a MERGE, first writes the literal string "Name: " to the output file. Then, after the user exits field NAME, the special GET destination OA$MERGE writes the contents of NAME to the file. Thus, if a user enters FROEHLE in field NAME, the output file contains Name: FROEHLE after the user leaves the form.

Interactively:

```
<OA$MRG_TTOUT\MERGE_LINE This is form <OA$FORM_NAME>
```

The call to OA$MRG_TTOUT directs MERGE_LINE output to the terminal, and the subsequent MERGE_LINE call processes the text with its single directive. If you do not precede MERGE_LINE with the call to OA$MRG_TTOUT, MERGE_LINE does nothing (except in the uncommon case when the currently displayed menu is called from within a template).

In a template:

```
<&OA MERGE_LINE Just like background text>
```

Calling MERGE_LINE from within a template is equivalent to simply including the text-line parameter as background text.

6.54.6 Special Features and Restrictions

Although ALL-IN-1 does not recognize symbols as MERGE_LINE parameters in the normal sense, MERGE translates any symbols you specify in angle brackets. Also, you can use GET OA$MERGE_LINE to build a MERGE_LINE function call with symbols. For example:

```
<GET OA$MERGE_LINE = NAME " is in department " PROFIL.DEPART[NAME]>
```

6.55 MERGE__LIST

6.55.1 Syntax

MERGE__LIST listfile__sym,formfile__sym,outfile__sym,specfile__sym
[,from][,to][,spcflag][,pflag][,aflag][,scrflag]

6.55.2 Description

The MERGE__LIST function calls WPS-PLUS List Processing to create an output file from input information. When WPS-PLUS is available under ALL-IN-1, you can use MERGE__LIST from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts

6.55.3 Function Parameters

MERGE__LIST takes three required parameters and seven optional ones:

listfile__sym

The first MERGE__LIST parameter is an ALL-IN-1 symbol that represents the name of a WPS-PLUS-formatted list file containing data elements used to create the output file. This file is analogous to, but more complex than, the optional list file processed by the MERGE function.

formfile__sym

The second parameter is an ALL-IN-1 symbol that represents the name of a WPS-PLUS-formatted form file. This file is analogous to the required template file processed by the MERGE function.

outfile_sym

The third parameter is an ALL-IN-1 symbol that represents the name of an output file that is created in WPS-PLUS format, using the information provided by the other files and flags. The outfile is analogous to the outfile file created by the MERGE function.

specfile_sym

The fourth, optional, parameter is an ALL-IN-1 symbol that represents the name of a WPS-PLUS-formatted specification file. This file contains instructions on combining a subset of the list file elements with the directives of the form file. When you omit specfile_sym, MERGE_LIST processes all elements in the list file.

You can call the TEST_SPEC function to test the specification file instructions before you call MERGE_LIST.

from

The fifth parameter, which is optional, is a positive number that corresponds to the position of the first list-file record, or entry, to process. When omitted, it defaults to 1.

to

The sixth parameter, which is optional, is a nonnegative number that corresponds to the position of the last list-file record, or entry, to process. When omitted, it defaults to 0, indicating that all records after the one indicated by the from parameter are processed.

The following three flags determine how MERGE_LIST modifies an existing output file.

spcflag

The seventh parameter, which is optional, is a supersede flag. When this parameter is 1 and you have not supplied any other flag parameters, MERGE_LIST creates a new output file that supersedes any preceding version of the file. When you omit this parameter or you supply a value other than 0, MERGE_LIST assumes a default of 1. When you supply a spcflag value of 0, MERGE_LIST overwrites any existing version of the file, guided by the values, if any, of the other two flag parameters, pflag and aflag.

pflag

The eighth parameter, which is optional, is a prefix flag. When this parameter is 1, MERGE_LIST inserts its output at the beginning of the specified output file, if it exists. If the file does not exist, MERGE_LIST creates it.

When you omit this parameter or when you supply a value other than 1, MERGE_LIST assumes a default of 0.

aflag

The ninth parameter, which is optional, is an append flag. When this parameter is 1, MERGE_LIST appends its output at the end of the specified output file, if it exists. If the file does not exist, MERGE_LIST creates it.

When you omit this parameter or you supply a value other than 1, MERGE_LIST assumes a default of 0.

NOTE: If more than one of aflag, pflag or spcflag is specified as a 1, then the order of precedence is: appending, prefixing, superseding.

scrflag

The tenth parameter, which is optional, is a screen flag. When this parameter is 1, MERGE_LIST displays the developing output on the terminal screen.

When you omit this parameter or when you supply a value other than 1, MERGE_LIST assumes a default of 0.

6.55.4 Function Qualifiers

None.

6.55.5 Examples

In a command procedure:

```
$ WRITE OAMAILBOX "OA MERGE_LIST #WPLFILE, #WPFFILE, #WPLOFILE,-"
$ WRITE OAMAILBOX "OA #WPLSFILE, ''frst_rec', ''last_rec',-"
$ WRITE OAMAILBOX "OA ''sprcede', ''pre', ''app', 1"
$ @DCLMAILBOX:
```

In this example, the command procedure passes the file-name parameters as local symbols and the optional flags as DCL symbols and a single string literal.

6.55.6 Special Features and Restrictions

MERGE_LIST is only valid if WPS-PLUS is linked with ALL-IN-1 during installation, as covered in the *Installation Guide*. The *WPS-PLUS/ALL-IN-1 User's Guide* covers this word processing system in detail.

6.56 MODE

6.56.1 Syntax

MODE [mode]

6.56.2 Description

MODE assigns the current Terminal Mode to one of four settings. The specified parameter, or lack of one, determines the mode, and you remain in that mode until you specify another mode.

You call MODE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.56.3 Function Parameters

mode

The single, optional MODE parameter is the mode to switch to. When you omit the parameter, the mode switches to Screen, the default mode.

The four MODEs are:

MODE A

Forces ALL-IN-1 to assume that a terminal is a VT100, even if it is not. This can be useful in cases where the terminal is only slightly incompatible with the VT100. MODE A is only useful if FMS can recognize the terminal you are on.

MODE C

Causes ALL-IN-1 to enter Hard Copy or Command Mode. This is the default if the terminal is unknown to ALL-IN-1. MODE C is useful when running ALL-IN-1 through a remote, dial-in line set at 1200 baud or in other situations where performance is a major factor.

Appendix B discusses Hard Copy Mode.

MODE S

Invokes Screen Mode, the ALL-IN-1 default Terminal Mode.

MODE V

Displays the current form name and form type on line 24 of the terminal. You can press Gold H while in MODE V and the current keywords in Help display on line 24. This mode is useful when debugging forms and command procedures and when building help files.

6.56.4 Function Qualifiers

None.

6.56.5 Examples

Interactively:

```
<PORT_ON\MODE C
```

The PORT_ON function is called interactively, followed by MODE C. ALL-IN-1 runs in Hard Copy Mode, and ALL-IN-1 sends the output of the LIST and PRINT_SCREEN functions to the attached printer.

```
<MODE V
```

Calling MODE V sets Verify Mode interactively to display form names on line 24. A second call to MODE V turns the mode off.

In Named Data:

Named Data

```
1 Name SCR
MODE S
```

Menu option SCR calls MODE S, which sets the terminal to Screen Mode.

In a command procedure:

```
$! Called while running in Hard Copy Mode to see if we can
$! run in "Approximate" Mode.
$!
$ WRITE OAMAILBOX "OA GET TERM__TYPE = OA$TERMINAL"
$ @DCLMAILBOX:
$ IF "'F$EXTRACT(0,2,TERM__TYPE)'" .NES. "VT" THEN EXIT
$!
$! This is a VT-series terminal; reset to Screen Mode:
$!
$ WRITE OAMAILBOX "OA MODE S"
$ @DCLMAILBOX:
```

6.56.6 Special Features and Restrictions

In earlier versions of ALL-IN-1, MODE N displayed the Named Data of the current form. Named Data now displays through view form OA$VIEW__ND, which is called by default by Gold N. Additional background information displays through view form OA$VIEW__FIELDS, which is called by default by Gold V.

Appendix B provides a list of terminal types that ALL-IN-1 recognizes.

6.57 NEWDIR

6.57.1 Syntax

NEWDIR [username]

6.57.2 Description

NEWDIR sets a user's default File Cabinet and ALL-IN-1 directory to those of another user.

You can call NEWDIR from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.57.3 Function Parameters

username

The single, optional parameter is the username of the user whose File Cabinet and ALL-IN-1 directory become the new defaults. NEWDIR obtains the value of the new default directory from the DIRECT field of the specified user's User Profile entry.

When you omit username, NEWDIR resets the File Cabinet and directory to the current user's defaults.

6.57.4 Function Qualifiers

None.

6.57.5 Examples

Interactively:

```
<NEWDIR BURKE
```

NEWDIR sets the default File Cabinet and directory to that of user BURKE.

In a command procedure:

```
$ WRITE OAMAILBOX "OA GET $TEMP=$USER"
$ @DCLMAILBOX:
$ WRITE OAMAILBOX "OA NEWDIR ''TEMP'"
$ @DCLMAILBOX:
```

6.57.6 Special Features and Restrictions

Although NEWDIR does not recognize symbols as parameters, you can use GET OA$FUNCTION to build NEWDIR calls from symbols or symbol lists.

NEWDIR resets the default directory, and allows you to create and modify documents that are stored in another user's File Cabinet, but the function does not reassign logical OAUSER. Therefore, if you attempt to run an ALL-IN-1 function that uses the logical OAUSER (for example, the CABINET subfunction REORG_DOCDB, which reorganises File Cabinets), it will have unpredictable results.

6.58 NEWLIB

6.58.1 Syntax

NEWLIB [library]

6.58.2 Description

NEWLIB closes and opens form libraries. When NEWLIB is called, it first does a DUMP_CACHE, causing all unused forms and files to be closed. If a form is still open when NEWLIB is called, then the form library containing those forms which are still in use remains open temporarily until all forms referencing that specific version of the library are closed. Any forms opened after the call to NEWLIB, come from the new version of the form library.

You call NEWLIB from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.58.3 Function Parameters

library

The single, optional NEWLIB parameter is the name of a single form library to close and then reopen. When you specify library, NEWLIB closes that library, flushes cached forms, and then opens the library again.

To increase performance, the NEWLIB function checks to see if there is a newer version of a form library before it closes and reopens a library. If only one form library has been changed, then only that library is reopened.

6.58.4 Function Qualifiers

None.

6.58.5 Examples

In Named Data:

Named Data

```
1 Name RESET
FORM PROFIL \ IFEXIT \ NEWLIB
```

Menu option RESET displays form set PROFIL, which accesses OA$DATA:PROFILE.DAT. If the user exits normally, ALL-IN-1 calls the NEWLIB function with no parameter to open any form libraries that have been added.

Interactively:

```
<NEWLIB USER
```

NEWLIB closes form library USER, flushes cached forms, and reopens USER.

In a command procedure:

```
$  ! Edit a form through FMS
$
$  ASSIGN/USER NLA0: SYS$OUTPUT
$  FMS USER/FORM = 'FORM
.
.
.
$  ! Call NEWLIB to bring up the latest version of the form.
$  !
$  ! $ WRITE OAMAILBOX "OA NEWLIB ''FRMLIB'"
$  @DCLMAILBOX:
```

6.58.6 Special Features and Restrictions

NEWLIB clears the prior-menu stack. Therefore if you press EXIT SCREEN (or call OA$FLD_EXIT) after calling NEWLIB, the previous menu does not display. However, NEWLIB establishes a new prior-menu stack.

If NEWLIB is specifically called on the form library which contains the current default form, NEWLIB does not close that form as long as the usage count is greater than zero. The library remains temporarily open until a SET_MENU function is called and a new form becomes the default.

NEWLIB does not affect the MEMRES form library in any way.

See also the DUMP_CACHE, OA$FLO_OPEN_LIB, and OA$FLO_CLOSE_LIB functions in this chapter for more information about opening and closing form libraries.

6.59 NEXT__LIST

6.59.1 Syntax

NEXT__LIST file-name-symbol

6.59.2 Description

NEXT__LIST returns the value of the next entry from a selection list file and loads two symbols with this value. The selected entry then cycles from the top of the list to the bottom.

You call NEXT__LIST from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.59.3 Function Parameters

file-name

The single, required NEXT__LIST parameter is the name of a selection-list file. NEXT__LIST stores the results of the function in the ALL-IN-1 permanent symbol $NEXT__LIST__ITEM and in the global DCL symbol NEXT__LIST__ITEM in the subprocess.

6.59.4 Function Qualifiers

None.

6.59.5 Examples

Interactively:

```
<NEXT__LIST ALTERNATE.SEL
```

NEXT__LIST stores the top entry in ALTERNATE.SEL in symbol $NEXT__LIST__ITEM.

In Named Data:

```
.PERIOD
NEXT__LIST WPLIST.SEL\GET @OA$FIELD__NAME=

.PERIOD
CAB$:DOCUMENT.TITLE[$NEXT__LIST__ITEM]
```

This Named Data keypad directive defines the keypad period to load the next item in a selection list into the current field. This definition lets you use the NEXT LIST key to fill in a field that is not qualified by the /LIST qualifier.

6.59.6 Special Features and Restrictions

For performance reasons, selection lists are usually kept in cache. The file is only updated on the disk when absolutely necessary. Therefore, if the selection list file is accessed by anything other than the NEXT__LIST or OA$FLD__NEXT__LIST functions, the file must be dumped from cache using the DUMP__CACHE function.

6.60 OA$CLI__GET__SYMBOL

6.60.1 Syntax

OA$CLI__GET__SYMBOL vms-symbol

6.60.2 Description

This function displays the translation of the specified VMS symbol on the terminal. This function calls the LIB$GET__SYMBOL facility in the VMS Run Time Library.

You can call OA$CLI__GET__SYMBOL from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.60.3 Function Parameters

vms-symbol

This single required parameter specifies the VMS symbol to be translated and displayed.

6.60.4 Function Qualifiers

None.

6.60.5 Examples

```
<OA$CLI__GET__SYMBOL BOZO
```

displays on line 24:

```
LIBR/REPLACE WRKD2$:[ARTHRAX.BUILD]DEVEL.OLB
```

6.60.6 Special Features and Restrictions

See VAX/VMS documentation set for more information on VMS symbols.

6.61 OA$CLI__GET__VALUE

6.61.1 Syntax

OA$CLI__GET__VALUE vms-parameter

6.61.2 Description

The OA$CLI__GET__VALUE function displays the value of the specified VMS (DCL) parameter on line 24.

You can call OA$CLI__GET__VALUE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.61.3 Function Parameters

vms-parameter

This single required parameter specifies the VMS parameter whose value is to be displayed. This parameter must be one from those that are active in the command, or command procedure, by which ALL-IN-1 was invoked.

6.61.4 Function Qualifiers

None.

6.61.5 Examples

```
<OA$CLI__GET__VALUE P1
```

returns the first parameter in a DCL command-line invocation.

6.61.6 Special Features and Restrictions

OA$CLI__GET__VALUE calls the VMS CLI$GET__VALUE function. See the VAX/VMS documentation set for more information.

6.62 OA$CNV__CONVERT

6.62.1 Syntax

OA$CNV__CONVERT from__form, to__form

6.62.2 Description

OA$CNV__CONVERT is a generic conversion function that takes as parameters the names of any two entry forms and copies the records from one associated data file to the other. This function:

- Copies all fields with the same name from one file to another
- Ignores fields from the old file for which there is no identically named field in the new file
- Puts the FMS default value into fields in the new file for which there is no identically named field in the old file

Fields of different sizes may be copied from the old file to the new, but the fields must have the same name.

You call OA$CNV__CONVERT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.62.3 Function Parameters

from_form

The first, required parameter is the name of a form that contains a .FILE directive which identifies a data file from which to copy.

to_form

The second, required parameter is the name of a form that contains a .FILE directive which identifies a destination data file for the copy.

6.62.4 Function Qualifiers

None.

6.62.5 Examples

Interactively:

```
<OA$CNV_CONVERT OLDAIE,NEWAIE
```

This call copies the contents of the data file associated with form OLDAIE to the file associated with form NEWAIE.

6.62.6 Special Features and Restrictions

To convert data files in which you have renamed fields, use the FOR function with WRITE as a DO subfunction.

6.63 OA$CNV__DIST__LIST

6.63.1 Syntax

OA$CNV__DIST__LIST v1__file,v2__file[,destination]

6.63.2 Description

The OA$CNV__DIST__LIST function converts a CMI distribution list from ALL-IN-1 V1.X format to V2.X format. In Version 1.X, a distribution list only contains the address of each user. In Version 2.X, each entry in the distribution list contains the ALL-IN-1 user's full name (from the User Profile) as well as the address.

You can call OA$CNV__DIST__LIST from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.63.3 Function Parameters

v1__file

The first parameter is the file name of the V1.X distribution list.

v2__file

The second parameter is the file name of the V2.X distribution list to create.

destination

The third, optional, parameter specifies a message router destination. When this parameter is specified, all special addresses in the V1.X distribution list are converted to message router addresses.

6.63.4 Examples

Assume a distribution list (in Version 1.X), in the file NAMES.TXT, contains the following names:

SMITH
NODE::JONES
@USERS
HARVEY

If the following function executes:

```
OA$CNV__DIST__LIST NAMES.TXT,NEWNAMES.TXT
```

A file, NEWNAMES.TXT, is created and it contains the following:

Harry Smith (SMITH)
Remote Addressee (NODE::JONES)
Mary Harvey (HARVEY)

If the function executes as:

```
OA$CNV__DIST__LIST NAMES.TXT,NEWNAMES.TXT,MRGATE
```

A file, NEWNAMES.TXT is created and it contains the following:

Harry Smith (SMITH)
Remote Addressee (JONES@MRGATE@NODE)
Mary Harvey (HARVEY)

6.63.5 Special Features and Restrictions

The DECMAIL distribution list may not be converted.

This function does not convert the distribution list header information. In Version 1.X, distribution lists are in the User Profile. In Version 2.X they are documents in the File Cabinet.

6.64 OA$CNV__MERGE

6.64.1 Syntax

OA$CNV__MERGE from__form, to__form

6.64.2 Description

OA$CNV__MERGE is a generic conversion function that takes as parameters the names of any two entry forms and copies unique records from one associated data file to the other. This function:

- Copies all records which are not already present in the to__form
- Copies all fields with the same name from one file to another
- Ignores fields from the old file for which there is not an identically named field in the new file
- Puts the FMS default value into fields in the new file for which there is not an identically named field in the old file

Fields of different sizes may be copied from the old file to the new, but the fields must have the same name.

This function differs from OA$CNV__CONVERT in that it does not create a new file. Instead it merges the contents of the from__file with the contents of the to__file.

You can call OA$CNV__MERGE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.64.3 Function Parameters

from_form

The first, required parameter is the name of a form that contains a .FILE directive which identifies a form to copy from.

to_form

The second, required parameter is the name of a form that contains a .FILE directive which identifies a form to copy to.

6.64.4 Function Qualifiers

None.

6.64.5 Examples

Interactively:

```
<OA$CNV_MERGE OLDAIE,NEWAIE
```

This call copies the records of the data file associated with form OLDAIE that are not already in the file associated with form NEWAIE to the file associated with form NEWAIE.

6.64.6 Special Features and Restrictions

To convert data files in which you have renamed fields, use the FOR function with WRITE as a DO subfunction.

6.65 OA$CNV__FROM__SCROLL

6.65.1 Syntax

OA$CNV__FROM__SCROLL indexed-file [,sequential-file]

6.65.2 Description

Use the OA$CNV__FROM__SCROLL function to convert an indexed file into a sequential file.

6.65.3 Function Parameters

indexed-file

This parameter is the name of the file in /SCROLL compatible format.

sequential-file

This optional parameter is the name of the sequential file. If you omit this parameter OA$CNV__FROM__SCROLL assumes that sequential-file has the same name as indexed-file.

6.65.4 Function Qualifiers

None.

6.66 OA$CNV__TO__SCROLL

6.66.1 Syntax

OA$CNV__TO__SCROLL sequential-file [,indexed-file]

6.66.2 Description

Use the OA$CNV__TO__SCROLL function to convert a sequential file into a simple indexed format compatible with the /SCROLL field qualifier. Once you have converted a file, you can display it in a scrolled region by identifying it with /SCROLL.

6.66.3 Function Parameters

sequential-file

This parameter is the name of the file to convert.

indexed-file

This optional parameter is the name of the file to create in /SCROLL-compatible format. If you omit this parameter OA$CNV__TO__SCROLL assumes that indexed-file has the same name as sequential-file.

6.66.4 Function Qualifiers

None.

6.67 OA$DSA__CACHE__STATUS

6.67.1 Syntax

OA$DSA__CACHE__STATUS

6.67.2 Description

ALL-IN-1 keeps track of recently or frequently accessed forms and data sets and caches them, or keeps them open to improve performance. Call OA$DSA__CACHE__STATUS to view which forms and data sets are currently cached in memory.

You call OA$DSA__CACHE__STATUS from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.67.3 Function Parameters

None.

6.67.4 Function Qualifiers

None.

6.67.5 Examples

Interactively:

```
<OA$DSA__CACHE
```

This interactive call displays the names of currently cached forms and data sets.

6.67.6 Special Features and Restrictions

Appendix C covers data sets in more detail and explains the statistics that display when you call OA$DSA__CACHE__STATUS.

6.68 OA$ENT__SET__xxx

6.68.1 Syntax

OA$ENT__SET__xxx

6.68.2 Description

These functions set the operation mode for the current entry form. They are useful when an ENTRY form is invoked without the operation mode being set (as for the USER PROFILE form). When such a form is invoked, a prompt asks the user to select the Operating Mode after selecting a record. The form that makes this prompt specifies in Named Data which OA$ENT__SET__xxx function to call in response to the user response. For example, OA$ENT__SET__INQUIRE is called when the user enters I. The functions are

- OA$ENT__SET__ADD
- OA$ENT__SET__CHANGE
- OA$ENT__SET__COPY
- OA$ENT__SET__DELETE
- OA$ENT__SET__INQUIRE

You can call OA$ENT__SET__xxx from within Named Data.

6.68.3 Function Parameters

None.

6.68.4 Function Qualifiers

None.

6.68.5 Special Features and Restrictions

OA$ENT__SET__xxx functions are not intended for general use by applications.

These functions are only useful for modifying the interface to the Operation Mode selection prompt for ENTRY forms. This interface is currently the form "ENTRYMENU".

6.69 OA$FBT_INSTALL_LIBRARY

6.69.1 Syntax

OA$FBT_INSTALL_LIBRARY compiled-form-library

6.69.2 Description

This function creates a permanent global section which contains the specified compiled form library file. (See the OA$FBT_WRITE_LIBRARY function for information on creating a compiled form library.) You usually call this function during system start-up to cause a single, shareable copy of the compiled form library to be resident in memory.

This function is optional. If you do not use it on a form library, each ALL-IN-1 process maintains in memory, a unique copy of compiled forms from that library. For smaller form libraries, this is usually not a problem. Form libraries that are in development or are changing frequently should not be installed.

You can call OA$FBT_INSTALL_LIBRARY from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.69.3 Function Parameters

compiled-form-library

The single parameter is the name of a compiled form library file whose default file extension is .FLC.

6.69.4 Function Qualifiers

None.

6.69.5 Examples

The following commands are usually submitted in a batch job during system start-up:

```
$ ALLIN1/NOINIT
Enter CMD:OA$FBT_INSTALL_LIB OALIB:MEMRES
Enter CMD:OA$FBT_INSTALL_LIB OALIB:OAFORM
Enter CMD:EXIT
```

6.69.6 Special Features and Restrictions

To use this function you need SYSGBL and PRMGBL privileges.

The function uses global pages. The number of pages it uses depends on the size of the .FLC file. Use the VMS INSTALL Utility /G option to list the number of remaining global pages. If there are not enough, you increase the number of pages using SYSGEN. Each installed compiled form library occupies one global section and thus reduces the number of available global sections by one.

The name of the section is the same as the file name (without directory and extension). Thus, it is impossible to install two libraries from different directories with the same name. However, this restriction does not mean that two different ALL-IN-1 systems can not run at the same time. When ALL-IN-1 references the global section, it determines if the section is the correct one. If it is not, then the .FLC is mapped privately. Thus, two ALL-IN-1 systems can run at the same time, but only one can have a given .FLC installed.

6.70 OA$FBT__LIST__LIBTREE

6.70.1 Syntax

OA$FBT__LIST__LIBTREE

6.70.2 Description

OA$FBT__LIST__LIBTREE lists the currently open form libraries. The function displays statistics about each library.

You call OA$FBT__LIST__LIBTREE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- From within validation expressions

6.70.3 Function Parameters

None.

6.70.4 Function Qualifiers

None.

6.70.5 Examples

Interactively:

```
<OA$FBT__LIST__LIBTREE
```

This interactive call displays the names of currently open form libraries in the following format:

```
FORM LIBRARY NAME
--.FLB open=n .FLC open=n Forms=XX Ref count=n
```

where

.FLB is the form library.

.FLC are the compiled ALL-IN-1 forms.

n is 0 if the library is closed and 1 if the library is open.

Forms = XX represents the number of forms in the library.

Ref count = n represents the number of forms which are in use.

The format is repeated for each of the form libraries.

6.71 OA$FBT__LIST__TREE

6.71.1 Syntax

OA$FBT__LIST__TREE [library-name]

6.71.2 Description

OA$FBT__LIST__TREE lists each form library name and the names of all forms in the form library.

You call OA$FBT__LIST__TREE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.71.3 Function Parameters

None.

6.71.4 Function Qualifiers

library-name

When this parameter is specified, only the forms for that library are listed.

6.71.5 Examples

Interactively:

```
<OA$FBT__LIST__TREE
```

This interactive call displays the names of currently open form libraries as well as the form names within each form library.

```
FORM LIBRARY NAME
--.FLB open=n .FLC open=n Forms=XX Ref count=n
FORM NAME
FORM NAME
.
.
.
```

where

.FLB is the form library.

.FLC are the compiled ALL-IN-1 forms.

n is 0 if the library is closed and 1 if the library is open.

Forms=XX represents the number of forms in the library.

Ref count=n represents the number of forms which are in use.

FORM NAME is the name of the form within the specified form library.

The format is repeated for each of the form libraries.

6.72 OA$FBT__REMOVE__LIBRARY

6.72.1 Syntax

OA$FBT__REMOVE__LIBRARY compiled-form-library

6.72.2 Description

This function removes from memory the global section that contains the specified compiled form library file. In effect, it reverses the OA$FBT__INSTALL__LIBRARY function. However, the reversal is not complete until all users who are referencing the form library exit from ALL-IN-1. Any user who opens a form library after this function is called, gets a privately mapped copy of the form library.

You can call OA$FBT__REMOVE__LIBRARY from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.72.3 Function Parameters

compiled-form-library

The single parameter is the file name of the precompiled form library to remove. The function uses only the file name, and not the extension.

6.72.4 Function Qualifiers

None.

6.72.5 Examples

```
OA$FBT__REMOVE__LIBRARY MEMRES
OA$FBT__REMOVE__LIBRARY OALIB:MEMRES
OA$FBT__REMOVE__LIBRARY OALIB:MEMRES.FLC
```

Each of these examples is equivalent.

6.72.6 Special Features and Restrictions

The function requires SYSGBL and PRMGBL privileges.

6.73 OA$FBT__WRITE__LIBRARY

6.73.1 Syntax

OA$FBT__WRITE__LIBRARY library-name

6.73.2 Description

This function creates a formatted file that contains a precompiled image of all forms in an FMS form library. If there is no .FLC file present for a given form library, then compilation is done using the .FLB file. The function is primarily for system manager's use. After a form library is stable, they may use this function to increase performance.

The function creates an output file (.FLC file extension) and puts it in the directory that contains the input file (.FLB extension). Both the input and the output file have the same name and must remain in the same directory where the .FLC file is created.

You can call OA$FBT__WRITE__LIBRARY from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.73.3 Function Parameters

library-name

The single parameter is the file name of a form library currently open by ALL-IN-1. The default extension is .FLB.

6.73.4 Function Qualifiers

None.

6.73.5 Examples

```
OA$FBT_WRITE_LIBRARY OALIB:OAFORM
```

The input file is OALIB:OAFORM.FLB and the output file is OALIB:OAFORM.FLC.

The memory resident form library (which is linked into the ALL-IN-1 image) may be compiled with the following command:

```
OA$FBT_WRITE_LIBRARY OALIB:MEMRES
```

The function stores the output as OALIB:MEMRES.FLC

6.73.6 Special Features and Restrictions

There are several restrictions on the OA$FBT_WRITE_LIBRARY function:

- The user needs write access to the directory which contains the .FLB file to compile.
- ALL-IN-1 uses the .FLC file until the form library is opened again. This occurs when you leave ALL-IN-1 and then return or if a NEWLIB function is called.
- You must open the form library before compiling it. This may be done with the OA$FLO_OPEN_LIB function.

See also the OA$FBT_INSTALL_LIBRARY function for information on making the .FLC file globally shareable.

6.74 OA$FLD__nnn

6.74.1 Syntax

OA$FLD__nnn

6.74.2 Description

OA$FLD__nnn functions are internal terminator functions that move the cursor between fields on the current form or onto the next or previous forms. These functions set dispositions that indicate whether you want to stay on the current form or field or move to the next field or form.

The FORM and FIELD functions call terminator functions internally when you press certain keys. Also, most terminator functions are associated, in the Named Data of form DEFAULT, with those same keys.

You call these functions from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

However, each function is only meaningful under given conditions. The following section covers how you use these functions.

Terminator functions take no parameters or qualifiers.

6.74.3 Terminator Functions

OA$FLD__BOTTOM

This function moves the cursor to the last field in the current form or form set. It sets the stay disposition by assigning a value of 3 to special symbol OA$FORM__DISPOSE.

This function is only meaningful when the current form or form set has more than one enterable field and when the user is in some field other than the last one.

OA$FLD_DONE

This function signals a successful exit from the current form or prompt. It sets the done disposition by assigning a value of 2 to special symbol OA$FORM_DISPOSE.

This function is only meaningful when there is a current form, form set, or prompt from which to exit.

OA$FLD_DOWN

This function moves the cursor to the first enterable field on the next line of the current form. It sets the stay disposition by assigning a value of 3 to special symbol OA$FORM_DISPOSE.

This function is only meaningful when the current form or form set has more than one enterable field and when you are in some field other than the last one. It is most useful when called from entry forms, like PROFIL, that contain several fields on a single line.

OA$FLD_EXIT

This function signals an immediate exit from the current form or prompt. It sets the exit disposition by assigning a value of 0 to special symbol OA$FORM_DISPOSE.

This function is only meaningful when there is a current form, form set, or prompt from which to exit.

OA$FLD_NEXT

This function moves the cursor to the next enterable field on the current form. It sets the stay disposition by assigning a value of 3 to special symbol OA$FORM_DISPOSE.

This function is only meaningful when the current form has more than one enterable field and when you are in some field other than the last one.

OA$FLD__PAGE__BACK

This function moves the cursor to the last enterable field on the previous page of a form set. It sets the stay disposition by assigning a value of 3 to special symbol OA$FORM__DISPOSE.

This function does not cause unpredictable behavior when called on a single-page form. However, it is only meaningful when the current form has more than one page.

OA$FLD__PAGE__BOTTOM

This function moves the cursor to the last enterable field on the current page of a form set. It sets the stay disposition by assigning a value of 3 to special symbol OA$FORM__DISPOSE.

OA$FLD__PAGE__FORWARD

This function moves the cursor to the first enterable field on the next page of a form set. It sets the stay disposition by assigning a value of 3 to special symbol OA$FORM__DISPOSE.

This function is only meaningful when called from a form that has more than one page.

OA$FLD__PAGE__TOP

This function moves the cursor to the first enterable field on the current page of a form set. It sets the stay disposition by assigning a value of 3 to special symbol OA$FORM__DISPOSE.

OA$FLD__PREV

This function moves the cursor to the previous enterable field on the current form. It sets the stay disposition by assigning a value of 3 to special symbol OA$FORM__DISPOSE.

This function is only meaningful when the current form or form set has more than one enterable field and when you are in some field other than the first one.

OA$FLD_STAY

This function sets the disposition of the form to the stay (value of 3) disposition. It is useful for overriding the user's terminator. The most common use of this function is to reset the disposition following a YESNO_PROMPT function call.

OA$FLD_TOP

This function moves the cursor to the first field in the current form or form set. It sets the stay disposition by assigning a value of 3 to special symbol OA$FORM_DISPOSE.

This function is only meaningful when the current form or form set has more than one enterable field and when you are in some field other than the first one.

OA$FLD_UP

This function moves the cursor to the first enterable field on the previous line of the current form. It sets the stay disposition by assigning a value of 3 to special symbol OA$FORM_DISPOSE.

This function is only meaningful when the current form or form set has more than one enterable field and when you are in some field other than the first one.

6.74.4 Examples

Interactively:

```
<PROMPT\OA$FLD_DONE
```

The PROMPT function displays the following on line 24: `Press RETURN to continue`. Regardless of what terminator the user presses at this message, OA$FLD_DONE will override its disposition and signal DONE.

```
<WAIT\OA$FLD_EXIT
```

The WAIT function displays `Press RETURN to continue` in the CHOICE field. Regardless of what terminator the user presses at this message, OA$FLD_EXIT signals EXIT.

In Named Data:

```
1 Name .Gold B
OA$FLD__BOTTOM

2 Name .DOWN
OA$FLD__DOWN

3 Name .TAB
OA$FLD__NEXT

4 Name .GOLD BACKSPACE
OA$FLD__PAGE__BACK

5 Name .GOLD TAB
OA$FLD__PAGE__FORWARD

6 Name .BACKSPACE
OA$FLD__PREV

7 Name .GOLD T
OA$FLD__TOP

8 Name .UP
OA$FLD__UP

9 Name .GOLD UP
OA$FLD__PAGE__TOP

10 NAME .GOLD DOWN
OA$FLD__PAGE__BOTTOM
```

Keypad directives associate terminator functions with key sequences in the Named Data of form DEFAULT.

6.75 OA$FLD__NEXT__LIST

6.75.1 Syntax

OA$FLD__NEXT__LIST

6.75.2 Description

The OA$FLD__NEXT__LIST function returns the value of the next entry from a selection list file. You can only call this function when the current field is qualified with the /LIST qualifier, which identifies the selection list to use. OA$FLD__NEXT__LIST then loads the next entry into two symbols ($NEXT__LIST__ITEM and NEXT__LIST__ITEM) and also into the current field.

You can call OA$FLD__NEXT__LIST from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.75.3 Function Parameters

None.

6.75.4 Function Qualifiers

None.

6.76 OA$FLD__RECOGNITION

6.76.1 Syntax

OA$FLD__RECOGNITION

6.76.2 Description

The OA$FLD__RECOGNITION function defines a key to perform recognition.

You can call OA$FLD__RECOGNITION from within Named Data.

6.76.3 Function Parameters

None.

6.76.4 Examples

In Named Data:

Named Data

```
1 Name .GOLD L
OA$FLD__RECOGNITION
```

6.76.5 Special Features and Restrictions

Form DEFAULT defines the Gold L key to call the OA$FLD__RECOGNITION function.

This function is only meaningful when a field is a key field or has been qualified with a /RSE__RECOGNITION, or /RECOGNITION field qualifier. If these conditions are not satisfied, a message appears on the screen and indicates that recognition is not available on the field.

6.77 OA$FLD__SCROLL__OFF

6.77.1 Syntax

OA$FLD__SCROLL__OFF

6.77.2 Description

This function turns off the internal flag which indicates that the user is in a scrolled region. The OA$FLD__nnn functions have special meanings when called from a scrolled region. OA$FLD__SCROLL__OFF allows you to call the OA$FLD__nnn functions without invoking these special features.

You can call OA$FLD__SCROLL__OFF from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.77.3 Function Parameters

None.

6.77.4 Examples

In Named Data:

Named Data

```
1 Name .GOLD F
OA$FLD__SCROLL__OFF\OA$FLD__DONE
```

In this example, the Gold F key is defined to perform the OA$FLD__DONE function without invoking the function's special features.

6.78 OA$FLD__SEARCH

6.78.1 Syntax

OA$FLD__SEARCH

6.78.2 Description

The OA$FLD__SEARCH function defines a key to perform a nonanchored recognition. The field must be either a key field on an Entry form, or be qualified with a /RECOGNITION field qualifier. If these conditions are not met, a message appears on the screen, and indicates that recognition is not available on the field.

You can call OA$FLD__SEARCH from within Named Data.

6.78.3 Function Parameters

None.

6.78.4 Examples

In Named Data:

Named Data

```
1 Name .GOLD S
OA$FLD__SEARCH
```

6.78.5 Special Features and Restrictions

Form DEFAULT defines the Gold S key to call the OA$FLD__SEARCH function.

The function acts like the OA$FLD__RECOGNITION function when the field is qualified by the /RSE__RECOGNITION qualifier.

You must have the PRVSRC privilege in your User Profile to use this function.

6.79 OA$FLO__CLEAR__VA

6.79.1 Syntax

OA$FLO__CLEAR__VA

6.79.2 Description

OA$FLO__CLEAR__VA clears the video attributes currently in effect (for example, turns off bolding if that is on). In particular, this function informs FMS to reset video attributes to those defined for its current context (for example what field it is on.) The normal ALL-IN-1 form processing and message display does this automatically.

You can call OA$FLO__CLEAR__VA from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.79.3 Function Parameters

None.

6.79.4 Function Qualifiers

None.

6.79.5 Special Features and Restrictions

This function is of use only in the case where the application (for example, a script) has output to the terminal causing video attributes to be set or changed.

6.80 OA$FLO__CLOSE__LIB

6.80.1 Syntax

OA$FLO__CLOSE__LIB form__library

6.80.2 Description

OA$FLO__CLOSE__LIB closes the specified form library. You call OA$FLO__CLOSE__LIB from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.80.3 Function Parameters

form__library

The single, required OA$FLO__CLOSE__LIB parameter is the form library to close. The function recognizes a full RMS file specification, including any logicals defined in the main process. The default file-type extension for the form__library parameter is .FLB, and OA$FLO__CLOSE__LIB appends this extension if you omit the file type. You must specify at least a form-library name.

6.80.4 Function Qualifiers

None.

6.80.5 Special Features and Restrictions

You typically call OA$FLO_CLOSE_LIB before extracting and/or modifying a form in a library. Always call this function with caution. In particular:

- Always call DUMP_CACHE either immediately before or after you call OA$FLO_CLOSE_LIB. If you do not, forms stored in the closed library may remain in memory, and when you reopen the library, you may abort ALL-IN-1.
- Never call OA$FLO_CLOSE_LIB to close the library in which the current form is stored. Although ALL-IN-1 may not immediately abort, some subsequent action will probably cause the image to exit abnormally.

Calling the NEWLIB function with a parameter is similar to the function list:

```
OA$FLO_CLOSE_LIB library \DUMP_CACHE \OA$FLO_OPEN_LIB library
```

Use NEWLIB when you want to open a new version of a form library, as opposed to updating an existing library.

Also see the OA$FBT_LIST_LIBTREE function, which displays a list of currently open form libraries.

6.81 OA$FLO__OPEN__LIB

6.81.1 Syntax

OA$FLO__OPEN__LIB form__library

6.81.2 Description

OA$FLO__OPEN__LIB opens the specified form library and makes its forms available to ALL-IN-1. You call OA$FLO__OPEN__LIB from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.81.3 Function Parameters

form__library

The single, required OA$FLO__OPEN__LIB parameter is the form library to open. The function recognizes a full RMS file specification, including any logicals defined in the main process. The default file-type extension for the form__library parameter is .FLB, and OA$FLO__OPEN__LIB appends this extension if you omit the file type. You must specify at least a form library name.

6.81.4 Function Qualifiers

None.

6.81.5 Special Features and Restrictions

Call OA$FLO__OPEN__LIB to open a new form library or to reopen a form library that you have closed previously. Always call this function with caution. In particular, always call DUMP__CACHE before you call OA$FLO__OPEN__LIB. If you do not, old images of forms stored in the library may still be in memory, and when you call one of these forms the conflicting references will abort ALL-IN-1.

Calling the NEWLIB function with a parameter is similar to the function list:

```
OA$FLO__CLOSE__LIB library \DUMP__CACHE \OA$FLO__OPEN__LIB library
```

Use NEWLIB when you want to open a new version of a form library, as opposed to updating an existing library.

See also the OA$FBT__LIST__LIBTREE function, which displays a list of currently open form libraries.

6.82 OA$FRM__SET__FIELD

6.82.1 Syntax

OA$FRM__SET__FIELD field-name

6.82.2 Description

OA$FRM__SET__FIELD is used to position the cursor on a specified field.

You can call OA$FRM__SET__FIELD from the following locations:

- In Named Data
- Interactively
- In scripts

6.82.3 Function Parameters

field__name

This required parameter specifies the name of the FMS field on the current form to make current.

6.82.4 Function Qualifiers

None.

6.83 OA$HAR_CHECK_DECCRT

6.83.1 Syntax

OA$HAR_CHECK_DECCRT

6.83.2 Description

OA$HAR_CHECK_DECCRT checks the terminal type and sets Hard Copy Mode appropriately. Hard Copy Mode is set ON, if the terminal is not recognized as a DEC-CRT class terminal. Hard Copy Mode is set OFF, if the terminal is recognized as a DEC-CRT.

You can call OA$HAR_CHECK_DECCRT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.83.3 Function Parameters

None.

6.83.4 Function Qualifiers

None.

6.83.5 Special Features and Restrictions

Applications could use this function to ensure that ALL-IN-1 is in Screen Mode, if the terminal supports it, even if the user has selected Hard Copy Mode.

Remember that the terminal characteristics, to which the terminal is set, may be other than DEC-CRT class even if the terminal is capable of acting as a DEC-CRT.

6.84 This section has been removed

6.85 OA$INI__GLOBALS

6.85.1 Syntax

OA$INI__GLOBALS

6.85.2 Description

OA$INI__GLOBALS sets up current variables in ALL-IN-1 from the user's User Profile. This is done automatically at ALL-IN-1 startup, unless ALL-IN-1 has been invoked with the /NOINIT qualifier. If, during the session, changes are made to the user's User Profile the user or an application would call OA$INI__GLOBALS in order to put those changes into effect (for example, editor style, calendar and work-week) for that session.

You can call OA$INI__GLOBALS from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.85.3 Function Parameters

None.

6.85.4 Function Qualifiers

None.

6.86 OA$INI__INITIALIZE

6.86.1 Syntax

OA$INI__INITIALIZE

6.86.2 Description

OA$INI__INITIALIZE causes a full ALL-IN-1 initialization to be done, as is normally done at ALL-IN-1 startup. This initialization is suppressed by the /NOINIT qualifier; this is the main use of OA$INI__INITIALIZE.

You can call OA$INI__INITIALIZE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.86.3 Function Parameters

None.

6.86.4 Function Qualifiers

None.

6.86.5 Examples

To use OA$INI__INITIALIZE:

1. Run ALL-IN-1 with the /NOINIT qualifier.
2. Perform whatever commands are needed before initialization.
3. Call OA$INI__INITIALIZE to cause normal initialization.

Alternately, OA$INI__INITIALIZE can be used to cause a reinitialization while running ALL-IN-1, after the environment (for example, User Profile or terminal type) has been altered. See also OA$INI__GLOBALS (which is a subset of OA$INI__INITIALIZE).

6.87 OA$INI__LOGICALS

6.87.1 Syntax

OA$INI__LOGICALS

6.87.2 Description

OA$INI__LOGICALS establishes the location and value of various VMS logicals used by ALL-IN-1; notably OA$LIB and OA$DATA. This function also establishes the name of the node upon which the session is being run. OA$INI__LOGICALS is a subset of OA$INI__INITIALIZE.

You can call OA$INI__LOGICALS from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.87.3 Function Parameters

None.

6.87.4 Function Qualifiers

None.

6.88 OA$INI__OPEN__LIB

6.88.1 Syntax

OA$INI__OPEN__LIB

6.88.2 Description

OA$INI__OPEN__LIB

- Opens the forms libraries named in the User Profile of a specific user.
- Opens OA$LIB:OAFORM and OA$LIB:MEMRES.
- Closes libraries, that are not in the list, that were already open.
- Reopens any form libraries that have been changed.

Libraries are normally opened at ALL-IN-1 startup, unless the /NOINIT qualifier has been used in invoking ALL-IN-1. This function is a subset of OA$INI__INITIALIZE.

You can call OA$INI__OPEN__LIB from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.88.3 Function Parameters

None.

6.88.4 Function Qualifiers

None.

6.89 OA$INI__SET__DEFAULT__DIR

6.89.1 Syntax

OA$INI__SET__DEFAULT__DIR

6.89.2 Description

OA$INI__SET__DEFAULT__DIR sets (or resets) the current directory to the original default directory, as specified in the User Profile. If ALL-IN-1 was invoked with the /RESTORE qualifier, then this function causes the original pre-session directory default to be restored upon exiting ALL-IN-1 (rather than leaving the default ALL-IN-1 directory in force).

You can call OA$INI__SET__DEFAULT__DIR from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.89.3 Function Parameters

None.

6.89.4 Function Qualifiers

None.

6.90 OA$INI_SUBPROCESS

6.90.1 Syntax

OA$INI_SUBPROCESS

6.90.2 Description

OA$INI_SUBPROCESS initializes the subprocess (but does not invoke it). The subprocess is normally initialized on start-up, when ALL-IN-1 has been invoked with the /NODEFER qualifier. Otherwise the subprocess will be initialized when it is first invoked by entering $ as a menu choice, or, a call is made which needs to access the subprocess.

You can call OA$INI_SUBPROCESS from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.90.3 Function Parameters

None.

6.90.4 Function Qualifiers

None.

6.90.5 Examples

You can use this function in conjunction with initialization, if you intend to use the subprocess in the session, and prefer to take the overhead for initializing it during ALL-IN-1 start-up. You might also use this function when invoking ALL-IN-1 with /NOINIT and performing your own initialization. For example, if you were to run ALL-IN-1 without full initialization, merely in order to execute ALL-IN-1 functions, rather than with the interactive form interface, you could invoke OA$INI_SUBPROCESS to run command files in the subprocess.

6.91 OA$KEY__COMMAND

6.91.1 Syntax

OA$KEY__COMMAND

6.91.2 Description

The OA$KEY__COMMAND function places the user into ALL-IN-1 Command Mode. Calling the OA$KEY__COMMAND function is the same as entering ALL-IN-1 using the /NOINIT CLI qualifier. To return to the default ALL-IN-1 Input Mode, enter RETURN at the following prompt:

```
Enter Command:
```

You can call OA$KEY__COMMAND from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In scripts

6.91.3 Function Parameters

None.

6.91.4 Special Features and Restrictions

This function requires the PRVCMD privilege in the User Profile.

6.92 OA$LIST__xxx

6.92.1 Syntax

OA$LIST__xxx

6.92.2 Description

These functions are invoked by responses to the <LIST function's bottom-of-screen (and end-of-listing) query. They are invoked via the form OA$LIST. This form defines which keys or terminators call which function. See the OA$LIST form for the current mapping of terminators to list functions.

The functions are:

OA$LIST__EXIT	quit (terminate listing)
OA$LIST__LEFT	scroll left
OA$LIST__NOPAUSE	cease querying at screen/page-breaks
OA$LIST__PAGE__FORWARD	resume listing at top of next page
OA$LIST__PAGE__TOP	back up to top of this page
OA$LIST__PRINT	print-screen (screen-image to file)
OA$LIST__REFRESH	refresh screen with current list display
OA$LIST__RIGHT	scroll right
OA$LIST__TOP	rewind to start of file
OA$LIST__WIDTH	toggle 80/132 column Screen-Mode

You call OA$LIST__xxx only from within Named Data of the OA$LIST form.

6.92.3 Function Parameters

None.

6.92.4 Function Qualifiers

None.

6.92.5 Examples

Examples of Named Data on OA$LIST form:

```
.GOLD P      OA$LIST__PRINT
.UP          OA$LIST__PAGE__TOP
.GOLD UP     OA$LIST__TOP
```

6.92.6 Special Features and Restrictions

OA$LIST should be used *only* while the <LIST function is active, hence only on the form OA$LIST (which the <LIST function invokes).

6.93 OA$LOG__DISPLAY

6.93.1 Syntax

OA$LOG__DISPLAY log-file

6.93.2 Description

OA$LOG__DISPLAY displays the specified ALL-IN-1 generated log file on the terminal. The function interprets and displays the binary header information in each record, including time-stamp.

You can call OA$LOG__DISPLAY from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.93.3 Function Parameters

log__file

This single required parameter specifies the VMS file-name of the ALL-IN-1 log to be displayed.

6.93.4 Function Qualifiers

None.

6.93.5 Special Features and Restrictions

The lines output by the ALL-IN-1 TRACE mechanism (A1LOG specifier) to a file (A1TRACE.LOG) are similar to, but not compatible with, the format displayed by OA$LOG__DISPLAY. A1TRACE.LOG files cannot be correctly displayed by this function. Instead, use ALL-IN-1 <LIST function, <EDIT function, VMS "TYPE", or an editor from VMS.

The log file to display must be closed, because no sharing of the file is assumed. The function sends the data directly to the screen unless you assign SYS$OUTPUT to another device. Then, it sends the data to the specified output device.

6.94 OA$LOG__TO__FILE

6.94.1 Syntax

OA$LOG__TO__FILE log-file

6.94.2 Description

OA$LOG__TO__FILE starts ALL-IN-1 logging, and directs logging output to the specified file. This output can then be displayed using the <OA$LOG DISPLAY function.

OA$LOG__TO__FILE:

- Requires PRMMBX privilege to create the mailbox.
- Creates OA$LOGGER logical pointing to MAILBOX, to receive logging messages from other ALL-IN-1 processes.
- Never RETURNS.
- Should normally be called in a batch job with /NOINIT.

You can call OA$LOG__TO__FILE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.94.3 Function Parameters

log-file

This single required parameter specifies the VMS file-name of the file to be created and to receive the logging output.

6.94.4 Function Qualifiers

None.

6.94.5 Examples

```
1  $SET PROC/PRIV = PRMMBX
2  $ALLIN1/NOINIT
3  Enter command:  OA$LOG__TO__FILE MYFILE.LOG
```

To stop this function use a CTRL/Y or the VMS STOP command.

6.95 OA$MENU__LEVEL__PUSH

6.95.1 Syntax

OA$MENU__LEVEL__PUSH

6.95.2 Description

OA$MENU__LEVEL__PUSH creates a new (but empty) menu level. The next form invoked will belong to that new menu level (whether or not it has already been invoked, and is still active on a previous menu level).

The function is useful in these contexts:

- When you want to isolate a menu sequence from previous levels (that is, a new incarnation of menus). This means that the menus of prior menu levels are invisible. (Compare the /CAPTIVE form qualifier and the invocation of the Interrupt menu by pressing Gold I, which creates a new menu level.)
- When a script wants to invoke a menu form. Use the <FORM function. To keep the script synchronized with ALL-IN-1, you must call <OA$MENU__LEVEL__PUSH, before calling <FORM, to create a new menu level.

You call OA$MENU__LEVEL__PUSH from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

NOTE: *When calling this function interactively or from the subprocess, it is necessary to specify a menu form with the function call (for example, <OA$MENU LEVEL PUSH\FORM X) since empty menu levels are not created.*

6.95.3 Function Parameters

None.

6.95.4 Function Qualifiers

None.

6.96 OA$MENU__SHOW__STACK

6.96.1 Syntax

OA$MENU__SHOW__STACK

6.96.2 Description

OA$MENU__SHOW__STACK displays the Menu Stack. This is the stack of currently open menus on each menu level.

Generally, within a menu level only one non-overlay menu and its overlays are open at a given time. Menu levels are created or exited via various functions and form qualifiers (notably /CAPTIVE). The Menu Stack display includes:

- Form name
- DSAB address
- If form is /CAPTIVE (CAP)
- If form is /OVERLAY (OVL)
- If form is /ONCE (ONCE)

The function is useful while developing applications with a number of overlays and/or form transitions (one form calling others). It is also useful with functions that support the /ONCE, /CAPTIVE, or /OVERLAY qualifiers. These qualifiers have a considerable effect on form flow behavior in ALL-IN-1, and it is often helpful to see the state of the menu stack after using them.

The primary purpose of this function is to debug the flow forms.

You call OA$MENU__SHOW__STACK from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In scripts
- Within validation expressions

6.96.3 Function Parameters

None.

6.96.4 Function Qualifiers

None.

6.97 OA$MRG__TTOUT

6.97.1 Syntax

OA$MRG__TTOUT [OFF]

6.97.2 Description

OA$MRG__TTOUT sends the results of all subsequent undirected MERGEs and MERGE__LINEs to the terminal until you call OA$MRG__TTOUT OFF. This function is useful for testing MERGE functionality interactively.

An undirected MERGE or MERGE__LINE is any call to these functions in which the output is not directed to a file. For example, if you call OA$MRG__TTOUT and then call either MERGE without the second (output-file) parameter or MERGE__LINE while MERGE is not currently active, the output is directed to the terminal.

You call OA$MRG__TTOUT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.97.3 Function Parameters

OFF

The single, optional OA$MRG__TTOUT parameter is the keyword OFF, which indicates an end to OA$MRG__TTOUT action. When you call this function with no parameters, OA$MRG__TTOUT directs otherwise undirected MERGE and MERGE__LINE output to the terminal. When you call OA$MERGE__TTOUT with a parameter, MERGE and MERGE__LINE once again direct their output to a specified output file.

6.97.4 Examples

Interactively:

```
<OA$MRG__TTOUT \MERGE__LINE Today is <OA$DATE> \OA$MRG__TTOUT OFF
```

This string of function calls displays the current date on the terminal.

6.98 OA$MSG__PURGE

6.98.1 Syntax

OA$MSG__PURGE

6.98.2 Description

ALL-IN-1 usually maintains a stack of messages to display. The OA$MSG__PURGE function removes any messages that are on the stack.

You can call OA$MSG__PURGE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.98.3 Function Parameters

None.

6.98.4 Examples

In Named Data:

Named Data

```
1 Name SEND
MAIL SEND\OA$MSG__PURGE\DISPLAY The message has been sent
```

6.99 OA$MSG__TRAP

6.99.1 Syntax

OA$MSG__TRAP

6.99.2 Description

Calling OA$MSG__TRAP causes ALL-IN-1 to trap severe errors (such as Access Violation) and signals them as messages on the screen, rather than crashing ALL-IN-1. This is not a permanent solution, to severe errors, but an aid to diagnosing them in ALL-IN-1 subsystems.

You can call OA$MSG__TRAP from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.99.3 Function Parameters

None.

6.99.4 Function Qualifiers

None.

6.100 OA$SCL__nnn

6.100.1 Syntax

OA$SCL__nnn

6.100.2 Description

OA$SCL__nnn functions are internal scrolling functions that you call to control processing in a scrolled region on a form. Scrolled regions are single field areas on a form that you create with FMS when editing or creating a form. The /SCROLL qualifier identifies a scrolled region to ALL-IN-1.

You call the OA$SCL__nnn functions from within Named Data.

6.100.3 Scroll Functions

OA$SCL__ALL [n]

This function allows the user to scroll a specified number of lines through the use of the parameter specified by n. For example: OA$SCL__ALL 10 scrolls ten lines at one time.

OA$SCL__BOTTOM

This function moves the cursor to the bottom of the scrolled region.

OA$SCL__DOWN

This function scrolls down one line.

OA$SCL__EXIT

This function exits the scrolled region and closes the scroll DSAB.

OA$SCL__FIRST__PAGE

This function brings the user the first page in the scrolled file and positions the cursor at the top of the scrolled region.

OA$SCL__INIT [max-line-no], [file-name], [start-line-number] [DSAB]

This function associates several keys with the other OA$SCL__nnn functions listed here. The keyboard definitions remain in effect until you call OA$SCL__EXIT.

The current scrolling information is set through four parameters.

where

max-line-no	is the maximum number of lines (records) in the scroll file. The limit is 2000 and is the default if this argument is omitted.
file-name	is an ALL-IN-1 symbol representing an indexed file.
start-line-number	is the starting record number that is loaded into the scrolled region. When omitted, start defaults to 1.
DSBA	can be any valid data set with two fields.

All of these arguments are positional, and you must insert placekeeping commas if you omit one or more arguments.

OA$SCL__LAST__PAGE

This function brings the user to the last page in the scrolled file and positions the cursor at the bottom of the scrolled region.

OA$SCL__NEXT__PAGE

This function brings the user to the next page in the scrolled file and positions the cursor at the top of the scrolled region.

OA$SCL__PRIOR__PAGE

This function brings the user to the previous page in the scrolled file and positions the cursor at the bottom of the scrolled region.

OA$SCL_RETURN

This function positions the cursor at the next line in a scrolled region, with two exceptions. If the cursor is positioned at the last line of data in the scrolled region, the function positions the cursor at the beginning of the next enterable field. If the cursor is positioned at the last line of data and there are no more enterable fields, the function signals a successful exit from the current form.

OA$SCL_SET_FIELD [max-line-no], [file-name], [start-line-no], [DSAB] [,field-name]

This function allows the user to navigate in a display-only scrolled region. The current scrolling information is set through five parameters.

where

max-line-no is the maximum number of lines (records) in the scroll file. The limit is 2000 and is the default if this argument is omitted.

file-name is an ALL-IN-1 symbol representing an indexed file.

start-line-no is the starting record number that is loaded into the scrolled region. When omitted, start defaults to 1.

DSAB can be any valid data set with two fields.

field-name is the name of the display-only field.

All of these arguments are positional, and you must insert placekeeping commas if you omit one or more arguments. The /SCROLL qualifier in Chapter 4 tells you how to use a display-only scrolled region.

OA$SCL_TOP

This function moves the cursor to the top of the scrolled region.

OA$SCL_UP

This function scrolls up one line.

6.100.4 File Conversion for /SCROLL

Use the OA$CNV_TO_SCROLL function to convert a sequential file into a simple indexed format compatible with the /SCROLL field qualifier. Once you have converted a file, you can display it in a scrolled region by identifying it with /SCROLL.

Use the OA$CNV_FROM_SCROLL function to convert an indexed file into a sequential file.

See the OA$CNV_TO_SCROLL and the OA$CNV_FROM_SCROLL functions in this chapter for more information.

6.100.5 Special Features and Restrictions

OA$SCL_nnn functions (with the exception of OA$SCL_INIT and OA$SCL_SET_FIELD) are meaningful when called while the current field is a scrolled region on a form, or when they follow the OA$SCL_SET_FIELD function. The ALL-IN-1 /SCROLL field qualifier calls these functions internally.

6.101 OA$SCP__STACK__DUMP

6.101.1 Syntax

OA$SCP__STACK__DUMP

6.101.2 Description

OA$SCP__STACK__DUMP displays the stacks of current <SCRIPT scripts and <DO scripts (in two separate lists). The function shows:

- Script name (as invoked by <DO or <SCRIPT function)
- Current line number in the script
- Script file (RMS file spec) from which the script was read

The stack results when scripts are nested (one script called while another script is active).

The intended use of this function is as a debugging tool.

You call OA$SCP__STACK__DUMP from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.101.3 Function Parameters

None.

6.101.4 Function Qualifiers

None.

6.102 OA$STAT_GET_STATISTICS

6.102.1 Syntax

OA$STAT_GET_STATISTICS

6.102.2 Description

OA$STAT_GET_STATISTICS snapshots various VMS job and process statistics for the ALL-IN-1 session's process. These statistics are then accessible through the following ALL-IN-1 symbols:

STAT$ASTCNT
STAT$ASTLM
STAT$BIOCNT
STAT$BIOLM
STAT$BUFIO
STAT$BYTCNT
STAT$BYTLM
STAT$CPULIM
STAT$CPUTIM
STAT$DFWSCNT
STAT$DIOCNT
STAT$DIOLM
STAT$DIRIO
STAT$FILCNT
STAT$FILLM
STAT$GPGCNT
STAT$JOBCNT
STAT$LOGINTIM
STAT$PAGFLTS
STAT$PID
STAT$PPGCNT
STAT$PRCCNT
STAT$PRI
STAT$PRIB
STAT$TQCNT
STAT$TQLM
STAT$UIC

STAT$xxx refers, in general, to the VMS statistic "xxx". For example, STAT$TQLM returns the Timer Queue Limit statistic. See Chapter 8 in this APR or the VAX/VMS documentation set ($GETJPI system service) for the meaning of each of these statistics.

You can call OA$STAT__GET__STATISTICS from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.102.3 Function Parameters

None.

6.102.4 Function Qualifiers

None.

6.102.5 Special Features and Restrictions

It is necessary to call this function before reading any of the statistics' symbols.

6.103 OA$SUB__CLOSE

6.103.1 Syntax

OA$SUB__CLOSE

6.103.2 Description

OA$SUB__CLOSE closes the subprocess if it has been created and is running. Usually, if the subprocess has been created, it is left open, in case it is needed again. OA$SUB__CLOSE can be used to free the system resources taken up by the subprocess, if it is not likely to be used again for some time.

You call OA$SUB__CLOSE from the following locations:

- In Named Data
- Interactively
- In scripts

6.103.3 Function Parameters

None.

6.103.4 Function Qualifiers

None.

6.104 OA$SUB__OPEN

6.104.1 Syntax

OA$SUB__OPEN

6.104.2 Description

OA$SUB__OPEN opens (CREATES) the ALL-IN-1 subprocess. This is normally done automatically by ALL-IN-1 the first time that the subprocess is needed.

OA$SUB__OPEN allows you to force the creation of the subprocess at a particular time (for example, at system start-up), rather than waiting for ALL-IN-1 to CREATE the subprocess.

This can also be done via the /NODEFER qualifier when invoking ALL-IN-1.

You call OA$SUB__OPEN from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.104.3 Function Parameters

None.

6.104.4 Function Qualifiers

None.

6.105 OA$SYM__CLOSE

6.105.1 Syntax

OA$SYM__CLOSE

6.105.2 Description

OA$SYM__CLOSE closes the currently open permanent symbol table file. See OA$SYM__OPEN.

You call OA$SYM__CLOSE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.105.3 Function Parameters

None.

6.105.4 Function Qualifiers

None.

6.106 OA$SYM__GET__SYMBOL

6.106.1 Syntax

OA$SYM__GET__SYMBOL symbol-name [=value]

6.106.2 Description

OA$SYM__GET__SYMBOL is similar to the <GET function, except that <GET does not display the symbol-name.

- When a value is specified, that value is assigned to the target symbol (the symbol is created if it does not already exist).
- When no value is specified, the value of the symbol is displayed on line 24 of the terminal.

You call OA$SYM__GET__SYMBOL from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.106.3 Function Parameters

symbol-name

This required parameter specifies the target symbol to be displayed, or into which a value is to be placed

value

This optional parameter specifies the value to be placed into the target symbol. This value may be a quoted text string or an ALL-IN-1 symbol.

6.106.4 Function Qualifiers

None.

6.106.5 Examples

Assume #TEMP does not exist:

Input: <OA$SYM__GET__SYMBOL #TEMP

Response: Symbol "#TEMP" not found

Input: <OA$SYM__GET__SYMBOL #TEMP = "ABC"

Input: <OA$SYM__GET__SYMBOL #TEMP

Response: #TEMP = "ABC"

6.107 OA$SYM__MAKE__SYMBOL

6.107.1 Syntax

OA$SYM__MAKE__SYMBOL symbol-name [text]

6.107.2 Description

OA$SYM__MAKE__SYMBOL creates a special or temporary ALL-IN-1 symbol. The value of the symbol is null if no value is provided.

You call OA$SYM__MAKE__SYMBOL from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.107.3 Function Parameters

symbol-name

The required parameter specifies the name of the symbol to be created.

text

The optional parameter specifies the value to place in the target symbol.

6.107.4 Function Qualifiers

None.

6.107.5 Examples

Assume #XX does not exist:

Input: <OA$SYM__GET__SYMBOL #XX

Response: `Symbol "#XX" not found`

Input: <OA$SYM__MAKE__SYMBOL #XX

Input: <OA$SYM__GET__SYMBOL #XX

Response: `#XX = ""`

6.108 OA$SYM__OPEN

6.108.1 Syntax

OA$SYM__OPEN [filename]

6.108.2 Description

OA$SYM__OPEN opens a permanent symbol table-file. If a file name is supplied, that name is used as the file name. The default file-type is .PST. If the file cannot be found, an attempt is made to find a .TMP file (which was the file-type in version 1 of ALL-IN-1). If that file is found, rename it to .PST. If neither of these files can be found, a .PST file is created.

You call OA$SYM__OPEN from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.108.3 Function Parameters

filename

This optional parameter specifies the name of the VMS symbol table-file to be opened or created.

6.108.4 Function Qualifiers

None.

6.108.5 Special Features and Restrictions

When ALL-IN-1 is invoked with initialization, or when the OA$INI__INITIALIZE function is invoked, OA$SYM__OPEN is also performed.

Many standard ALL-IN-1 forms reference permanent symbols in the user's Permanent Symbol Table. Therefore, OA$SYM__OPEN must be performed before form processing of such forms.

6.109 OA$TRA_OUT

6.109.1 Syntax

OA$TRA_OUT text

6.109.2 Description

OA$TRA_OUT outputs a text line to the trace destination(s) currently in effect (see OA$TRA_SET function). This can be used by applications to annotate a trace.

You call OA$TRA_OUT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.109.3 Function Parameters

text

The single required parameter is the text to be output.

6.109.4 Function Qualifiers

None.

6.110 OA$TRA__SET

6.110.1 Syntax

OA$TRA__SET [specifiers]

6.110.2 Description

OA$TRA__SET turns on and off, tracing and tracing-output destinations for ALL-IN-1 components. The following list of specifiers allows you to choose what to trace, and where to put the trace output. OA$TRA__SET is especially useful in developing scripts, and in complex Named Data usage.

You call OA$TRA__SET from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.110.3 Function Parameters

The specifiers govern the extents of tracing, and the destination(s) of trace output. The syntax is

TRACE [OFF,] <specifier>

where <specifier> can be traceable functions or trace output destinations.

Traceable functions:

DPEINI	= trace DPE editor initialization commands
FORM	= trace opening and closing of each form
FIELD	= trace field-processing
FLOW	= trace form-processing
HELP	= trace help
INPUT	= trace (pseudo-)terminal input to A1
IVP	= trace for Installation Verification Procedures
MESS	= trace messages output to line 24
PUT	= trace puts to FMS form-fields
SCRIPT	= trace SCRIPT commands (see . SCRIPT ".TRACE" command)
SCRCHR	= trace each SCRIPT character returned to ALL-IN-1
SYMBOL	= trace symbol processing
A1LOG	= include A1-logging output (interleaved) in trace-log (superset of LOG)
ALL	= combination of all Flow-Control tracing

Destinations:

TERM	= output trace to terminal
LOG	= output trace to A1TRACE.LOG (preset to TRUE)

6.110.4 Function Qualifiers

None.

6.110.5 Examples

To trace SCRIPT commands and ALL-IN-1 Form Processing and form transitions, and display on the terminal while also logging to file A1TRACE.LOG, use:

```
OA$TRA_SET TERM,LOG,SCRIPT,FORM
```

To turn off the FORM tracing and the terminal display, but leave the logging of SCRIPT commands to A1TRACE.LOG, use:

```
OA$TRA_SET OFF,FORM,OFF,TERM
```

To record, in Hard Copy Mode, the entire sequence of user inputs, together with Hard Copy prompts and messages output by ALL-IN-1 (that is, a picture of the session), use:

```
OA$TRA_SET LOG,INPUT,MESS
```

6.110.6 Special Features and Restrictions

Trace functions may generate a lot of output. Make sure that the specified trace output really is required. Try to use a specific trace function rather than the /ALL qualifier, wherever possible.

6.111 OA$TRM__BRDCST__INIT

6.111.1 Syntax

OA$TRM__BRDCST__INIT

6.111.2 Description

The OA$TRM__BRDCST__INIT function enables broadcast-message trapping in ALL-IN-1. When you invoke OA$TRM__BRDCST__INIT the value of the symbol OA$BRDCST__TRAP is set to 1.

You call OA$TRM__BRDCST__INIT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.111.3 Function Parameters

None.

6.111.4 Function Qualifiers

None.

6.111.5 Examples

Interactively:

<OA$TRM__BRDCST__INIT

Through OAMAILBOX:

A command procedure to call an application might look like this:

```
$WRITE OAMAILBOX "OA GET #SAVE__TRAP=OA$BRDCST__TRAP"
$WRITE OAMAILBOX "OA IF #SAVE__TRAP=1 THEN OA$TRM__BRDCST__STOP"
$@DCLMAILBOX:
.
.
. DCL to run application
.
.
$WRITE OAMAILBOX"OA IF #SAVE__TRAP=1 THEN OA$TRM__BRDCST__INIT"
$@DCLMAILBOX:
```

6.111A OA$TRM__BRDCST__STOP

6.111A.1 Syntax

OA$TRM__BRDCST__STOP

6.111A.2 Description

The function OA$TRM__BRDCST__STOP turns off broadcast-message trapping. If you run ALL-IN-1 with broadcast-message trapping enabled, then you should use OA$TRM__BRDCST__STOP when you intend to call applications, such as DECgraph, which run in the subprocess and do their own broadcast-message trapping. If you do not disable ALL-IN-1 broadcast-message trapping, then you may encounter problems when attempting to use the application.

When you invoke OA$TRM__BRDCST__STOP the value of the symbol OA$BRDCST__TRAP is set to 0.

When the user returns to ALL-IN-1, you should ensure that he turns broadcast-message trapping back on (see OA$TRM__BRDCST__INIT).

You can call OA$TRM__BRDCST__STOP from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.111A.3 Function Parameters

None

6.111A.4 Function Qualifiers

None

6.111A.5 Examples

Interactively:

```
<OA$TRM__BRDCST__STOP
```

Through OAMAILBOX:

A command procedure to call an application might look like this:

```
$WRITE OAMAILBOX "OA GET #SAVE__TRAP=OA$BRDCST__TRAP"
$WRITE OAMAILBOX "OA IF #SAVE__TRAP=1 THEN OA$TRM__BRDCST__STOP"
$@DCLMAILBOX:
.
.
. DCL to run application
.
.
$WRITE OAMAILBOX"OA IF #SAVE__TRAP=1 THEN OA$TRM__BRDCST__INIT"
$@DCLMAILBOX:
```

6.112 OA$TRM__INIT

6.112.1 Syntax

OA$TRM__INIT

6.112.2 Description

OA$TRM__INIT initializes the terminal input and output channels, depending on the terminal's (or SYS$INPUT/SYS$OUTPUT) device characteristics as currently known to VMS. This function is always called at ALL-IN-1 start-up, whether or not /NOINIT has been specified. It is not generally used after that.

Its only use after this would be where the VMS terminal characteristics have been changed during the ALL-IN-1 session.

NOTE: *OA$TRM__INQUIRE also sets the VMS terminal characteristics to match what the terminal (or terminal device) has reported it as.*

You call OA$TRM__INIT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.112.3 Function Parameters

None.

6.112.4 Function Qualifiers

None.

6.113 OA$TRM__INQUIRE

6.113.1 Syntax

OA$TRM__INQUIRE

6.113.2 Description

OA$TRM__INQUIRE inquires of the terminal its characteristics, and saves these characteristics inside ALL-IN-1. This is not done during initialization, unless the /DEVICE__TYPE=INQUIRE qualifier is specified when invoking ALL-IN-1.

NOTE: *OA$TRM__INQUIRE sets the VMS terminal characteristics to match what the terminal (or terminal device) has reported it as.*

You call OA$TRM__INQUIRE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.113.3 Function Parameters

None.

6.113.4 Function Qualifiers

None.

6.114 OA$TRM__PORT__LINE

6.114.1 Syntax

OA$TRM__PORT__LINE text

6.114.2 Description

OA$TRM__PORT__LINE prints a line of text on the printer-port, if the port is currently enabled (OA$TRM__PORT__ON or OA$TRM__PORT__LOG__ON). This is similar to OA$TRM__PORT__PRINT, except that the port device is only enabled to output one line. Then it is disabled.

This is designed for applications which output a single line to the port, or interleave printer-port output with output to the terminal display.

You call OA$TRM__PORT__LINE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- From within validation expressions

6.114.3 Function Parameters

text

This required parameter specifies the text string to be output.

6.114.4 Function Qualifiers

None.

6.115 OA$TRM__PORT__LOG__ON

6.115.1 Syntax

OA$TRM__PORT__LOG__ON

6.115.2 Description

OA$TRM__PORT__LOG__ON enables ALL-IN-1 to output to the terminal printer-port, if one is attached to the terminal. This is similar to OA$TRM__PORT__ON; except that output is also echoed on the terminal screen. Use OA$TRM__PORT__OFF to discontinue output to the printer port.

You call OA$TRM__PORT__LOG__ON from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.115.3 Function Parameters

None.

6.115.4 Function Qualifiers

None.

6.116 OA$TRM__PORT__OFF

6.116.1 Syntax

OA$TRM__PORT__OFF

6.116.2 Description

OA$TRM__PORT__OFF disables ALL-IN-1 output to the terminal printer-port.

See OA$TRM__PORT__ON and OA$TRM__PORT__LOG__ON for how to enable output to the printer-port.

You call OA$TRM__PORT__OFF from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.116.3 Function Parameters

None.

6.116.4 Function Qualifiers

None.

6.117 OA$TRM__PORT__ON

6.117.1 Syntax

OA$TRM__PORT__ON

6.117.2 Description

OA$TRM__PORT__ON enables ALL-IN-1 output to the terminal printer-port, if one is attached to the terminal (see also the function: OA$TRM__PORT__LOG__ON). Output is not echoed to the terminal screen.

Use OA$TRM__PORT__OFF to disable the printer port.

You call OA$TRM__PORT__ON from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.117.3 Function Parameters

None.

6.117.4 Function Qualifiers

None.

6.118 OA$TRM__PORT__PRINT

6.118.1 Syntax

OA$TRM__PORT__PRINT text

6.118.2 Description

OA$TRM__PORT__PRINT prints a text line on the printer-port, if the port is currently enabled (OA$TRM__PORT__ON or OA$TRM__PORT__LOG__ON). This is similar to the OA$TRM__PORT__LINE function, except that the port is left enabled after the call (until OA$TRM__PORT__OFF is invoked).

This is designed for an application which outputs a series of lines and avoids the overhead of disabling and re-enabling the device for each line.

You call OA$TRM__PORT__PRINT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.118.3 Function Parameters

text

This single required parameter specifies the text string to be output.

6.118.4 Function Qualifiers

None.

6.119 OA$TXL__CLOSE

6.119.1 Syntax

OA$TXL__CLOSE

6.119.2 Description

The OA$TXL__CLOSE function closes an open TXL. You use this function with the OA$TXL__OPEN function if you update the TXL during an ALL-IN-1 session and want to start using the latest version of the TXL.

You can call OA$TXL__CLOSE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.119.3 Function Parameters

None.

6.119.4 Function Qualifiers

None.

6.119.5 Special Features and Restrictions

If the ALL-IN-1 session is accessing any element in the TXL at the time the OA$TXL__CLOSE function is called, the TXL is not closed immediately. Instead, it is closed when no element within the TXL is being accessed. For example, if a TXL script invokes the OA$TXL__CLOSE function, the script must exit before the TXL is closed.

6.120 OA$TXL__COMPILE

6.120.1 Syntax

OA$TXL__COMPILE

6.120.2 Description

The OA$TXL__COMPILE function compiles a new TXL from the VMS source directories (OA$BLP, OA$DO, and OA$SCP) for the various TXL elements. Note the following conditions when using the OA$TXL__COMPILE function:

- If a given directory does not exist or is empty, then the TXL does not contain elements of that type.
- If there is no TXL open, the newly-compiled TXL becomes the current one. Otherwise, to make the new TXL the current one, the previous one must be closed using the OA$TXL__CLOSE function.
- The TXL is created in the OA$LIB directory.

You can call OA$TXL__COMPILE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.120.3 Function Parameters

None.

6.120.4 Function Qualifiers

None.

6.121 OA$TXL__DIRECTORY

6.121.1 Syntax

OA$TXL__DIRECTORY

6.121.2 Description

The OA$TXL__DIRECTORY function displays a list of all elements in the current TXL.

You can call OA$TXL__DIRECTORY from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.121.3 Function Parameters

None.

6.121.4 Function Qualifiers

None.

6.122 OA$TXL__INSTALL

6.122.1 Syntax

OA$TXL__INSTALL

6.122.2 Description

The OA$TXL__INSTALL function installs the most recent TXL section-file as a global section. This section-file is the file created in the OA$LIB directory by using the OA$TXL__COMPILE function. See the *System Manager's Guide* for more information on installing TXLs.

You can call OA$TXL__INSTALL from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.122.3 Function Parameters

None.

6.122.4 Function Qualifiers

None.

6.122.5 Special Features and Restrictions

You must have PRMGBL and SYSGBL privileges to execute this function.

If a TXL is already installed, and you want to install a new TXL, then you must first remove the current one with the OA$TXL__REMOVE function.

6.123 OA$TXL__LIST

6.123.1 Syntax

OA$TXL__LIST element-name [,element-type]

6.123.2 Description

The OA$TXL__LIST function displays one element in the current TXL.

You can call OA$TXL__LIST from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.123.3 Function Parameters

element-name

This parameter is the name of a script or boilerplate in the current TXL.

element-type

This optional parameter is either SCRIPT, DO, or BLP. The default is SCRIPT.

6.123.4 Function Qualifiers

None.

6.123.5 Examples

```
<OA$TXL__LIST  WPPRINT,DO
```

6.124 OA$TXL__OPEN

6.124.1 Syntax

OA$TXL__OPEN

6.124.2 Description

The OA$TXL__OPEN function opens the shared global section TXL which was last installed. If none was installed, it opens the latest compiled TXL file. You do not usually need to use this function, since an attempt to access a TXL element (for example, with the SCRIPT, DO, MERGE, or OA$TXL__LIST functions) automatically causes the TXL to be opened.

You can call OA$TXL__OPEN from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.124.3 Function Parameters

None.

6.124.4 Function Qualifiers

None.

6.124.5 Special Features and Restrictions

The OA$TXL__OPEN function has no effect if there is a current TXL open. You must use the OA$TXL__CLOSE function to close the open TXL. Then the TXLs usage count must be at zero, to allow the OA$TXL__CLOSE function to take effect.

If you attempt to access a TXL element, and no TXL is found, ALL-IN-1 makes no additional attempts to look for a TXL. You must use the OA$TXL__OPEN function to make ALL-IN-1 look for a TXL again.

6.125 OA$TXL__REMOVE

6.125.1 Syntax

OA$TXL__REMOVE

6.125.2 Description

The OA$TXL__REMOVE function removes the TXL shared global section which is installed in memory. See the *System Manager's Guide* for more information.

You can call OA$TXL__REMOVE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.125.3 Function Parameters

None.

6.125.4 Function Qualifiers

None.

6.125.5 Special Features and Restrictions

You must have PRMGBL and SYSGBL privileges to execute this function.

6.126 OA$VAL__SET__VALID

6.126.1 Syntax

OA$VAL__SET__VALID

6.126.2 Description

Call OA$VAL__SET__VALID to indicate that the contents of the current field are valid: that is, that they meet some sort of validation criteria. The validation qualifiers /INVALID, /RECOGNITION, and /VALID usually call this function internally, but with some validation expressions specified with these qualifiers, you must call OA$VAL__SET__VALID explicitly.

OA$VAL__SET__VALID is only meaningful when called from within a validation expression.

6.126.3 Function Parameters

None.

6.126.4 Function Qualifiers

None.

6.126.5 Examples

In a validation expression:

Named Data

```
1 Name TERMINAL
/VALID= <.IF TERMINAL:2 .EQS. "VT" THEN OA$VAL__SET__VALID

2 TERMINAL
ELSE DISPLAY That is not a valid terminal type
```

/VALID validates the first two characters in field TERMINAL against the value "VT". If they match, the .FX directive calls OA$VAL__SET__VALID, and /VALID accepts the user input as valid.

6.127 PARSE_USER

6.127.1 Syntax

PARSE_USER symbol

6.127.2 Description

PARSE_USER parses an ALL-IN-1 symbol which is usually a username. The function extracts and concatenates the first nine (or less) characters from the symbol (omitting blanks, special characters and numerics) and saves this value in permanent symbol $PARSE_USER_NAME. See also the function MAKE_FILE_NAME.

You call PARSE_USER from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.127.3 Function Parameters

symbol

The required symbol parameter is an ALL-IN-1 symbol. PARSE_USER strips out all special characters, concatenates the first nine (or less) characters, and saves the parsed value in permanent symbol $PARSE_USER_NAME. When you call it from the subprocess, PARSE_USER also stores the parsed value in global DCL symbol PARSE_USER_NAME.

Unlike MAKE_FILE_NAME, PARSE_USER does not check existing files or attempt to create a unique file-name root.

6.127.4 Function Qualifiers

None.

6.127.5 Examples

In Named Data:

Named Data

```
1 Name USER
/POST = 'PARSE_USER USER

2 Name USER
\MAKE_FILE_NAME FILNAME, $PARSE_USER_NAME,

3 Name USER
 OA$DEFAULT_EXTENSION'
```

After the user leaves field USER, PARSE_USER parses the contents of that field, and MAKE_FILE_NAME creates a unique file name using $PARSE_USER_NAME and the user's default file-type extension, storing that name in field FILNAME.

Interactively:

```
<PARSE_USER "D'Angelo"
```

This interactive call stores DANGELO in $PARSE_USER_NAME.

In a command procedure:

```
$ ! Get the username, parse it, and use it as a file-name root.
$ !
$  WRITE OAMAILBOX "OA PARSE_USER OA$USER"
$ @DCLMAILBOX:
$ !
$ ! Nine or less characters returned in DCL symbol
$ ! PARSE_USER_NAME.
$ !
$ DIR/OUT = TEMP.LIS/COL = 1 'PARSE_USER_NAME'.*
```

6.128 PORT__ON, PORT__OFF

6.128.1 Syntax

PORT__ON|OFF

6.128.2 Description

The PORT__ON and PORT__OFF functions activate and deactivate, respectively, any printer port on your terminal. ALL-IN-1 does not direct terminal output immediately to the printer port. Instead, PORT__ON causes the LIST, PRINT__SCREEN, and in some cases, the MERGE function, to output to the printer port.

6.128.3 Function Parameters

None.

6.128.4 Function Qualifiers

None.

6.128.5 Examples

Interactively:

```
<PORT__ON \ LIST TEXTFILE.TXT \PORT__OFF
```

This interactive entry switches the printer port on and then outputs TEXTFILE.TXT to the printer. PORT__OFF switches the printer port off after the listing is complete.

6.128.6 Special Features and Restrictions

PORT__ON and PORT__OFF do not check whether the terminal has an attached printer port or, when it does, whether there is a printer hooked up.

See also the LIST and PRINT__SCREEN functions in this chapter.

6.129 PRINT__SCREEN

6.129.1 Syntax

PRINT__SCREEN [dest__file]

6.129.2 Description

The PRINT__SCREEN function makes a copy of the current screen (lines 1 thru 23). It prints only the FMS forms and not the overlaid CBI plots or editor screens. The destination of the output depends on the following conditions:

- PRINT__SCREEN outputs to an attached printer port if:
  - Your terminal has a printer port.
  - You have flagged the presence of the port by setting the PRINTER__PORT User Profile field to Y.
  - The port is active, that is, you have called the PORT__ON function.
- If you are currently in the text editor, then PRINT__SCREEN stores the screen image in the primary buffer.
- Otherwise, PRINT__SCREEN stores the output in the file you have identified as the function parameter, or in your scratch pad (file PASTE.TMP) if you have not specified a parameter. If the output file already exists in the user's directory, PRINT__SCREEN appends the screen image to it, separated by a form feed. You can then manipulate the output file like any other RMS text file.

You call PRINT__SCREEN from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.129.3 Function Parameters

dest__file

The single, optional parameter is the name of a destination file in which PRINT__SCREEN stores the screen image. PRINT__SCREEN ignores this parameter if you call the function while the printer port is active or while you are in the editor. When you omit the parameter, the filename defaults to PASTE.TMP.

6.129.4 Function Qualifiers

None.

6.129.5 Examples

Interactively:

```
<PRINT__SCREEN
```

Calling PRINT__SCREEN interactively produces a copy of the current menu.

6.130 PROMPT

6.130.1 Syntax

PROMPT [prompt-symbol]

6.130.2 Description

PROMPT displays text on line 24 and places the cursor on that line directly after the message. Control passes to the user, and the PROMPT remains active until the user presses RETURN, EXIT SCREEN, or some key sequence that has been explicitly associated with a field terminator. PROMPT saves any user response in DCL symbol $RESULT and special symbol OA$PROMPT_TEXT, and stores the terminator value in these symbols:

- DCL symbol TERMINATOR
- Special symbol OA$FORM_TERMINATOR
- Special symbol OA$FIELD_TERM

PROMPT stores identical values in TERMINATOR, OA$FORM_TERMINATOR, and OA$FIELD_TERM. It stores the form disposition in OA$FORM_DISPOSE. The value of the terminator determines the form disposition.

You call PROMPT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.130.3 Function Parameters

prompt-symbol

The single, optional PROMPT parameter is an ALL-IN-1 symbol that contains the text of the prompt. When you omit prompt-symbol, PROMPT displays the message: **Press RETURN to continue.** Like other ALL-IN-1 symbols, prompt-symbol may comprise a list of different symbol types.

6.130.4 Function Qualifiers

None.

6.130.5 Examples

Interactively:

```
<PROMPT
```

Calling PROMPT with no parameters from a menu displays the following message on line 24:

```
Press RETURN to continue
```

ALL-IN-1 suspends processing until the user presses RETURN or EXIT SCREEN.

```
<PROMPT "Is your name " OA$USER "?"
```

Two quoted strings and a special symbol are specified to build the prompt string.

In a command procedure:

```
$ WRITE OAMAILBOX "OA PROMPT ""Press RETURN or EXIT SCREEN"""
$ @DCLMAILBOX:
$ IF TERMINATOR .EQ. 112 THEN EXIT
$!
$! Continue processing...
```

PROMPT determines if the user wants to continue the procedure. If the user presses keypad 0, or EXIT SCREEN, then the TERMINATOR has a value of 112.

6.130.6 Special Features and Restrictions

Although prompts are just single-line requests at the bottom of a screen, they have some form-like features. Specifically, any Named Data keypad definition (for example, Gold H, Gold V, and so forth) that ALL-IN-1 recognizes in a field on a form will also be recognized at a prompt. Also PROMPT saves the user's response in the following symbols:

- OA$FORM_DISPOSE — Special ALL-IN-1 symbol. Stores the disposition of PROMPT function as an odd or even number.
- OA$FORM_TERMINATOR — Special ALL-IN-1 symbol. Stores the numeric value of the key that the user presses to terminate the PROMPT. Table 4-3 lists these values.
- OA$FIELD_TERM — Special ALL-IN-1 symbol. Stores the same numeric value as OA$FORM_TERMINATOR.
- OA$PROMPT_TEXT — Special ALL-IN-1 symbol. PROMPT loads this symbol with the value of any text the user enters on line 24 before pressing the terminator.
- TERMINATOR — Global DCL symbol. When you call PROMPT from the subprocess, it gives TERMINATOR the value of OA$FORM_TERMINATOR.
- RESULT — Global DCL symbol. PROMPT loads RESULT with the text the user enters on line 24.

If HELP (Gold H) is called from a prompt, ALL-IN-1 first looks at special symbol OA$PROMPT_HELP for a keyword list. If that symbol is empty, then HELP looks for topic, form PROMPT, where form is the form that is current when the prompt is called and PROMPT is a keyword corresponding to a level-2 topic under form.

OA$PROMPT_HELP is one of several special symbols and expressions that you can update with GET, for example:

```
<GET OA$PROMPT_HELP = "topic subtopic"
```

Since you make this assignment dynamically, you can provide different help for different prompts or even for the same prompt called at different times.

YESNO_PROMPT is similar to PROMPT but only recognizes yes, no, y, and n as valid replies. The WAIT function is also similar to prompt but displays a message at the current cursor position instead of line 24.

6.131 PROMPT__ID

6.131.1 Syntax

PROMPT__ID prompt-id [prompt-symbol]

6.131.2 Description

PROMPT__ID is the same as PROMPT except that the first word after the function name will be saved, for the duration of the prompt, as the prompt's name. This prompt name is stored in the ALL-IN-1 symbol OA$PROMPT__ID and is not displayed as part of the prompt.

If a SCRIPT Mode script is active when the prompt is invoked, the script can test OA$PROMPT__ID to identify the prompt. See PROMPT for more information. See chapter 8 for more information on OA$PROMPT__ID.

PROMPT__ID displays text on line 24 and places the cursor on that line directly after the message. Control passes to the user, and the PROMPT__ID remains active until the user presses RETURN, EXIT SCREEN, or some key sequence that has been explicitly associated with OA$FLD__DONE (RETURN) or OA$FLD__EXIT (EXIT SCREEN). PROMPT__ID saves any user response in DCL symbol $RESULT and special symbol OA$PROMPT__TEXT, and stores the terminator value in these symbols:

- DCL symbol TERMINATOR
- Special symbol OA$FORM__TERMINATOR
- Special symbol OA$FIELD__TERM

PROMPT__ID stores identical values in TERMINATOR, OA$FORM__TERMINATOR, and OA$FIELD__TERM. It stores the form disposition in OA$FORM__DISPOSE. The value of the terminator determines the form disposition.

You call PROMPT__ID from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.131.3 Function Parameters

prompt-id

This required parameter is the name of the prompt that is stored in OA$PROMPT__ID.

prompt-symbol

This optional PROMPT__ID parameter is an ALL-IN-1 symbol that contains the text of the prompt. When you omit prompt-symbol, PROMPT__ID displays the message: `Press RETURN to continue`. Like other ALL-IN-1 symbols, prompt-symbol may comprise a list of different symbol types.

6.131.4 Function Qualifiers

None.

6.132 PURGE__FILE

6.132.1 Syntax

PURGE__FILE file, keep-count [[,file,keep-count]...]

6.132.2 Description

The PURGE__FILE function purges the specified RMS file or files. The syntax and results of this function are similar to those of the VMS PURGE command, but you express the number of versions to keep as a parameter instead of through a qualifier.

You call PURGE__FILE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.132.3 Function Parameters

file...

You can specify one or more RMS file names; you must provide at least one. PURGE__FILE recognizes full RMS file specifications, including logical names that are defined in the context of the main process.

When you do not specify a directory, PURGE__FILE assumes that the file(s) are in your default ALL-IN-1 directory, OAUSER, rather than the current directory, if the two are different.

keep-count

You can optionally specify the number of versions to keep after each file specification. When you omit this parameter or when you specify an invalid value such as 0, keep-count defaults to 1.

Supply placekeeping commas if you omit a keep-count while purging multiple files.

6.132.4 Function Qualifiers

None.

6.132.5 Examples

In Named Data:

Named Data

```
1 Name PF
PURGE__FILE TEST.TXT\DISPLAY Test files purged\FORCE
```

When the user selects menu option PF, PURGE__FILE deletes all but the last version of TEST.TXT files in directory OAUSER. Then the message **Test files purged** is displayed.

Interactively:

```
<PURGE__FILE THISFILE.TXT,2,SYS$LOGIN:THATFILE.TMP,,OTHERFILE.LIS
```

PURGE__FILE deletes all but the last two versions of OAUSER:THISFILE.TXT and all but the last version of SYS$LOGIN:THATFILE.TMP and SYS$LOGIN:OTHERFILE.LIS.

In a command procedure:

```
$ ! Purge OAUSER, and confirm the operation
$ !
$ WRITE OAMAILBOX "OA FORM GETCOUNT\"
$ !
$ ! In PURGEALL's Named Data is a /PUT__SAVE field qualifier that
$ ! creates DCL symbol PURGE__COUNT from information supplied by
$ ! the user. Also, the background text prompts the user:
$ ! Press RETURN to purge your directory, EXIT SCREEN to exit.
$ !
$ WRITE OAMAILBOX "IFEXIT\PURGE__FILE OAUSER,''PURGE__COUNT'"
$ ! @DCLMAILBOX:
$ !
$EXIT
```

The following command procedure is equivalent, assuming that OAUSER has the same value in the subprocess that it has in the main process.

```
$ WRITE OAMAILBOX  "OA FORM GETCOUNT"
$ @DCLMAILBOX:
$ !
$ ! Test for EXIT SCREEN
$ !
$ IF TERMINATOR .EQ. 112 THEN EXIT
$ !
$ ! Delete the file
$ !
$ PURGE/KEEP = 'PURGE__COUNT' OAUSER
$ !
$EXIT
```

6.132.6 Special Features and Restrictions

Although PURGE__FILE does not recognize symbols as parameters, you use GET OA$FUNCTION to build PURGE__FILE calls from symbols or symbol lists.

Like DELETE__FILE, PURGE__FILE occurs in the main process and is thus subject to the privileges, default protections, logical name tables, and other characteristics of the main process, as opposed to those of the subprocess. Thus, if you alter privileges in the subprocess to gain delete

access to a protected file, then you cannot purge the file with PURGE_FILE; you must use the DCL PURGE command. For example, if file NEWDOCDB.DAT has the following protection:

```
(SYSTEM:RWED,OWNER:RWED,GROUP:RE,WORLD)
```

and you are not the owner of this file, then the following sequence from the subprocess does not work:

```
$ SET PROC/PRIV = SYSPRV
$ WRITE OAMAILBOX "OA PURGE_FILE NEWDOCDB.DAT"
  . . .
```

Similarly, PURGE_FILE does not recognize process logical names that have been assigned in the subprocess.

6.133 PUT_FIELD

6.133.1 Syntax

PUT_FIELD [field,]symbol

6.133.2 Description

PUT_FIELD puts value into a field on the current form. You call this function from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.133.3 Function Parameters

field

The first parameter, which is optional, is the FMS name of the field into which PUT_FIELD inserts the symbol value. The field may be any field on the form. PUT_FIELD with a field parameter is equivalent to:

GET field=symbol

When you omit the field name, PUT_FIELD processes the current field.

symbol

The second parameter is an ALL-IN-1 symbol to insert into the specified field. This symbol may be any valid symbol including another field on the same form.

6.133.4 Function Qualifiers

None.

6.133.5 Examples

In Named Data:

```
1    Name    USER
/VAL = PROFIL/SHOW = FULNAM/POST = 'PUT_FIELD STATUS,
PROFIL.STATUS[USER]'
```

When you enter field USER, ALL-IN-1 validates your entry against the User Profile. If you move off this field normally, the /POST qualifier calls PUT_FIELD to load field STATUS with the value of the STATUS field in the Profile.

6.133.6 Special Features and Restrictions

The PUT_FIELD function is similar to other functions and qualifiers, including GET, the /GET form qualifier, and the /GET_SAVE and /COPY field qualifiers.

You use PUT_FIELD to load a value into either a display-only field or an enterable field. However, if you call PUT_FIELD interactively from a menu, the value will only remain on the screen for a short time. Each time you enter something (such as an interactive function call) in the choice field and press RETURN, you cycle through the Form Processing loop illustrated in Figures 2-1 and 2-2. Since a call to PUT_FIELD does not display a new form, the current menu does not repaint, but its qualifiers are processed as if you had just called the menu for the first time. Thus, any changes you make to a menu field with PUT_FIELD are immediately overridden by any preprocessing qualifiers associated with that field.

6.134 QUEUE__BATCH

6.134.1 Syntax

QUEUE__BATCH filename__symbol[[/qual]...]

6.134.2 Description

QUEUE__BATCH sends a command procedure to a VMS batch queue with any instructions provided by the optional qualifiers. This function is similar to the DCL SUBMIT command, which is covered in the VAX/VMS documentation set.

You call QUEUE__BATCH from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts

6.134.3 Function Parameters

filename__symbol

The single, required QUEUE__BATCH parameter is an ALL-IN-1 symbol that represents the name of an ASCII file. DCL treats the file as a command procedure. You must specify at least a file-name root and file-type extension.

6.134.4 Function Qualifiers

You call QUEUE__BATCH with any combination of the following qualifiers. Specify all qualifiers after the file-name parameter.

NOTE: When you specify symbols as qualifier arguments that contain characters like " or :, parsing errors may result unless you enclose the symbol in parentheses. In particular, modify references such as $WPDOC:36:6 to be ($WPDOC:36:6).

/AFTER = after-time-sym

Use /AFTER to cause VMS to hold the batch job until the time specified by after-time-sym, the required argument.

When you omit /AFTER, QUEUE_BATCH submits the job for immediate processing.

/[NO]DELETE

Use /DELETE to delete a batch-job file after VMS processes it. The default setting is /NODELETE.

/[NO]IDENTIFY
/[NO]JOBMSG

Use these qualifiers interchangeably to notify the user that QUEUE_BATCH has placed the batch job in the specified queue. The message displays on line 24 in the format:

```
Job "(job name)" sent to queue (queue) as job (job number)
```

where

job name — is the file name by default. Use /NAME to specify another job name.

queue — is the default ALL-IN-1 batch queue. Use /QUEUE to specify another queue. The default ALL-IN-1 batch queue is specified in the OAGBL module. It is usually SYS$BATCH, but may be redefined during installation and customization at a particular site. See the System Manager for the specific queue in use at your site.

job number — is a sequential value that VMS assigns when the job enters the queue.

/[NO]KEEP

Use /KEEP in conjunction with /PRINT to override the default QUEUE__BATCH action of deleting the batch log file after VMS prints it.

/NAME = name-string-sym

Use /NAME to override the default QUEUE__BATCH action of giving the job the same name as the submitted batch file. The required name-string-sym argument is an ALL-IN-1 symbol that represents a one- to nine-alphanumeric-character string.

/[NO]PRINT

Use /PRINT to queue the batch log file to SYS$PRINT for printing when VMS finishes processing the batch job. The default is /NOPRINT.

/PARAMS = (param-syml,[param-sym2]...[,param-sym8])

Use /PARAMS to pass from one to eight optional DCL parameters to the batch job. Each parameter is an ALL-IN-1 symbol that corresponds to values for DCL symbols P1 through P8. If you specify more than one parameter, separate them by commas, and enclose the entire list in parentheses.

/PRIORITY = priority-sym

Use /PRIORITY to override the default QUEUE__BATCH action of submitting the batch job with the same priority as the base priority of your process. The required priority-sym argument is an ALL-IN-1 symbol that represents a numeral from 0 to 31. Jobs with larger priority values are processed quicker than those with smaller priority values.

When you use /PRIORITY to reduce a job's priority, it is usually to move large, noncritical jobs into the background of the batch stream. You may also use /PRIORITY to give a job a priority higher than its base priority, but doing so requires the VMS OPER privilege. Set this privilege in the main process before running ALL-IN-1 if you need to raise the priority of a batch job.

/QUEUE = queue-sym

Use /QUEUE to send the batch job to a specific batch queue. The required queue-sym parameter is an ALL-IN-1 symbol that represents the queue name. When you omit this qualifier, QUEUE_BATCH sends the job to the default ALL-IN-1 batch queue.

NOTE: ALL-IN-1 makes a single logical-name translation on queue-sym before QUEUE_BATCH submits the job.

6.134.5 Examples

From Named Data:

```
1 Name QB
DO BATCHPREP \ QUEUE_BATCH #BATCH_FILE /AFTER = #TIME  /PAR = "ALL"
```

Menu option QB calls DO to process script BATCHPREP.SCP and then QUEUE_BATCH to submit a batch job. Local symbol #BATCH_FILE identifies the job, which QUEUE_BATCH submits after a period of time equal to #time and with parameter P1 equal to ALL.

Interactively:

```
<QUEUE_BATCH  'DOC' /PARAMS = (#P1,#P2)/QUEUE = DOC$BATCH
```

This interactive call submits DOC.COM to queue DOC$BATCH (which must have been set up previously by the system manager). QUEUE_BATCH submits the job with two parameters: The values of local symbols #P1 and #P2.

```
<QUEUE_BATCH  'COMPLINK'/PARAMS = (#FILE, "Debug", OA$TIME)
```

This interactive call submits COMPLINK.COM with three parameters: P1 is the value of local symbol #FILE, P2 is Debug, and P3 is the value of special symbol OA$TIME.

6.134.6 Special Features and Restrictions

Like COMMAND, QUEUE_BATCH acts upon a command procedure. Thus, calling:

```
<COMMAND 'DOIT' "BUILD" "ALL"
```

is similar to calling:

```
<QUEUE__BATCH 'DOIT'/PAR=("BUILD", "ALL")
```

VMS processes the parameters identically in both cases. The main difference is that the COMMAND function executes DOIT.COM immediately, in the context of your ALL-IN-1 subprocess, while QUEUE__BATCH submits this command procedure to the batch queue and frees your terminal for other interactive tasks.

NOTE: Any time you submit a batch job with <COMMAND or <QUEUE__BATCH, VMS creates a detached process to execute the commands. This process receives your disk and directory defaults and the same quotas and privileges that you had when you logged in. Thus, if you submit a command procedure that does not reside in SYS$LOGIN, or one that sets default to another directory while it is executing, you must ensure that this procedure allows for any differences in file protections, privileges, UIC settings, and default file specifications.

See the SUBMIT command in the VAX/VMS documentation set for more information.

6.135 QUEUE_PRINT

6.135.1 Syntax

QUEUE_PRINT filename_symbol[[/qual]...]

6.135.2 Description

QUEUE_PRINT queues a file to a printer with any instructions provided by the optional qualifiers. This function is similar to the DCL PRINT command, which is covered in the VAX/VMS documentation set.

You can call QUEUE_PRINT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts

6.135.3 Function Parameters

filename_symbol

The single, required QUEUE_PRINT parameter is an ALL-IN-1 symbol that represents the name of an ASCII file to queue to a printer. You must specify at least a file-name root and file-type extension.

6.135.4 Function Qualifiers

You can call QUEUE_PRINT with any combination of the following 12 qualifiers. Specify all qualifiers after the file-name parameter.

NOTE: When you specify symbols as qualifier arguments that contain characters like " or :, parsing errors may result unless you enclose the symbol in parentheses. In particular, modify references such as $WPDOC:36:6 to be ($WPDOC:36:6).

/AFTER = after-time-sym

Use /AFTER to cause VMS to hold the print job until the time specified by after-tim-sym, the required argument.

When you omit /AFTER, QUEUE__PRINT submits the job for immediate printing.

/[NO]BURST

/BURST, the default setting, prints an extra banner page at the beginning of a listing that makes locating the print job easier when it is taken from the printer. Specify /NOBURST to output a single banner page.

/COPIES = count-sym

Use /COPIES to specify the number of copies to print. The required count-sym argument is an ALL-IN-1 symbol that represents a numeral from 1 to 255. QUEUE__PRINT prints one copy when you omit /COPIES.

/[NO]DELETE

Use /DELETE to delete a file after printing; the default setting is /NODELETE.

/[NO]FEED

Use /NOFEED to override the default printer action of automatically inserting form feeds when it nears the end of a page.

/[NO]FLAG__PAGE

Use /NOFLAG__PAGE to override the default printer action of automatically printing a flag or banner page preceding the output. QUEUE__PRINT prints the name of the job across the banner page. You can change the name of the job with the /NAME qualifier. — which is the file name by default, but which you can change with the /NAME qualifier — across the banner page.

/FORM = type__sym

Specifies the name of the form for the print job (for example "A4"). The required type__sym argument is an ALL-IN-1 symbol representing the form type.

/[NO]HEADER

Use /HEADER to print the name of the file at the top of each output page. /NOHEADER is the default.

/[NO]IDENTIFY
/[NO]JOBMSG

Use these qualifiers interchangeably to notify the user that QUEUE__PRINT has placed the print job in the specified queue. The message displays on line 24 in the format:

```
Job "⟨job name⟩" sent to queue ⟨queue⟩ as job ⟨job number⟩
```

where

job name — is the file name by default. Use /NAME to specify another job name.

queue — is the default ALL-IN-1 print queue. Use /QUEUE to specify another queue. The default ALL-IN-1 print queue is specified in the OAGBL module. It is usually SYS$PRINT, but may be redefined during installation and customization at a particular site. See the System Manager for the specific queue in use at your site.

job number — is a sequential value that VMS assigns when the job enters the queue.

/NAME = name-string-sym

Use /NAME to override the default QUEUE__PRINT action of giving the job the same name as the file being printed. The required name-string-sym argument is an ALL-IN-1 symbol that represents a one- to nine-|alphanumeric-|character string.

/[NO]NOTIFY

Use /NOTIFY to broadcast a message to the user's terminal when the print job has been completed or aborted. The default for this qualifier is set by the print completion flag in the User Profile.

/PRIORITY = priority-sym

Use /PRIORITY to override the default QUEUE__PRINT action of giving the print job the same priority as the base priority of your process. The required priority-sym argument is an ALL-IN-1 symbol that represents a numeral from 0 to 31. VMS prints jobs with larger priority values before those with smaller values.

When you use /PRIORITY to reduce a job's priority, it is usually to move large or low-priority jobs into the background of the print queue. For example, if you QUEUE__PRINT 20 files at priority 1, and the queue is empty, VMS starts printing the first file immediately; but as soon as someone else queues a print job with a higher priority, their job begins printing as soon as the current level-1 job completes. VMS thus prints low-priority jobs on an as-available basis.

You may also use /PRIORITY to give a print job a priority higher than its base priority, but doing so requires the VMS OPER privilege. Set this privilege in the main process before running ALL-IN-1 if you need to raise the priority of either a print or batch job.

/QUEUE = queue-sym

Use /QUEUE to send the print job to a specific print queue. The required queue-sym parameter is an ALL-IN-1 symbol that represents the queue name. When you omit this qualifier, QUEUE__PRINT sends the job to the default ALL-IN-1 print queue.

NOTE: ALL-IN-1 makes a single logical-name translation on queue-sym before QUEUE__PRINT submits the job.

6.135.5 Examples

From Named Data of form PHPRIN:

```
2 Name B
GET OA$FUNCTION="MERGE PH" $PH__SHORT__DIR "BRIEF.BLP,PH"
$PH__SHORT__DIR

3 Name B
"BRIEF.LIS,PH" $PH__SHORT__DIR "LIST.SEL"\GET OA$FUNCTION=

4 Name B
"QUEUE__PRINT PH" $PH__SHORT__DIR "BRIEF.LIS/DELETE/JOBMSG"
```

From Named Data of form AIPRIN:

```
2 Name B
MERGE OA$LIB:AIBRIEF.BLP,AIBRIEF.LIS,ACTITEM.SEL"

3 Name B
\QUEUE__PRINT AIBRIEF.LIS/DELETE/JOBMSG
```

In both cases, menu option B first calls the MERGE function to create an output file and then QUEUE__PRINT to print it. On form PHPRIN, QUEUE__PRINT is actually called by GET OA$FUNCTION, which builds the name of the output file from the same symbol list used to build the MERGE output file.

Interactively:

```
<QUEUE__PRINT MEMO.LIS /COPIES=5
```

This call sends five copies of MEMO.LIS to SYS$PRINT.

6.135.6 Special Features and Restrictions

See the VAX/VMS documentation set for information about the similar DCL PRINT command.

6.136 RENAME

6.136.1 Syntax

RENAME "current-file-name" "new-file-name"

6.136.2 Description

This function renames the specified VMS file to a new name. The names can be ALL-IN-1 symbols or quoted strings. RENAME is similar to the DCL RENAME command except that it does not handle wildcards.

You call RENAME from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- From within validation expressions

6.136.3 Function Parameters

current-file-name

This required parameter specifies the VMS file that you wish to rename.

new-file-name

This required parameter specifies the new VMS file name.

6.136.4 Function Qualifiers

None.

6.136.5 Special Features and Restrictions

See the VAX/VMS DCL documentation for further information on the RENAME command.

6.137 REPEAT

6.137.1 Syntax

REPEAT

6.137.2 Description

REPEAT causes a series of stacked functions that precede it to be repeated.

This function is only meaningful and useful at the end of a function list that contains at least one IFEXIT.

6.137.3 Function Parameters

None.

6.137.4 Function Qualifiers

None.

6.137.5 Examples

In Named Data:

Named Data

```
1 Name ABC
FORM ABC ARG/FIELDS=X,Y,Z\IFEXIT\REPEAT
```

The REPEAT call in this function list causes Form ABC to display and be processed as an argument form until the user presses EXIT SCREEN.

Interactively:

```
<FORM ABC \ IFEXIT \ FORM NEWARG \ REPEAT
```

Form ABC displays, followed by NEWARG if the user exits normally. The cycle is repeated until EXIT SCREEN is pressed in ABC.

6.137.6 Special Features and Restrictions

If IFEXIT is not included in the function list with REPEAT, an infinite loop is created. To stop an infinite loop use CTRL/C.

6.138 SAVE

6.138.1 Syntax

SAVE symbol [literal]

6.138.2 Description

The SAVE function creates and assigns values to ALL-IN-1 permanent, local, and updatable special symbols.

You call SAVE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.138.3 Function Parameters

symbol

The first SAVE parameter is the name of a symbol-table symbol. If the symbol does not already exist, SAVE creates it; otherwise, SAVE updates it.

If symbol is a permanent symbol, SAVE creates (or updates) a global DCL symbol and gives it the same value and name (less the "$" prefix) as symbol.

literal

The second parameter, which is optional, is the value SAVE assigns to the symbol, expressed as a literal string. When you omit literal, SAVE clears the symbol. The only restriction on literal is its length: 482 characters.

You may assign values that contain special characters such as quotes and slashes, but this is not recommended since these characters may be misinterpreted in other functions.

6.138.4 Function Qualifiers

None.

6.138.5 Examples

In Named Data:

```
1 Name JFIELD
/POST = 'SAVE #NVAL The current value is JANUS'
```

When you leave JFIELD, the /POST field qualifier calls SAVE, which assigns a value to local symbol #NVAL.

Interactively:

```
<SAVE $TFILE Temporary File for Testing
```

SAVE assigns permanent symbol $TFILE the following value: Temporary File for Testing. SAVE also assigns this value to global DCL symbol TFILE.

In a command procedure:

```
$ !
$ ! Get the file-name root of FILNAM
$ !
$ ROOT := = "'F$EXTRACT (0,F$LOCATE(".", FILNAM),FILNAM)'"
$ !
$ ! Save the value of ROOT in an ALL-IN-1 local symbol
$ !
$ WRITE OAMAILBOX "OA SAVE ''ROOT'"
$ @DCLMAILBOX:
```

6.138.6 Special Features and Restrictions

Although SAVE does not recognize symbols as parameters, you may use GET OA$FUNCTION to build SAVE calls from symbols or symbol lists.

SAVE symbol value

is functionally equivalent to the following GET function call:

```
<GET symbol = 'value'
```

6.139 SCRIPT

6.139.1 Syntax

SCRIPT script-file-name

6.139.2 Description

The SCRIPT function invokes the Script facility. SCRIPT is used to process indirect command files, or scripts, which are used as:

- Computer Based Instructions (CBIs)
- User Defined Proceses (UDPs)
- ALL-IN-1 system demonstrations
- Installation Verification Procedures (IVPs)

You call SCRIPT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.139.3 Function Parameters

script-file-name

The single, required parameter is the name of an ASCII file that contains SCRIPT commands and text. SCRIPT recognizes full RMS device and directory specifications. When you do not specify a device or directory, SCRIPT searches for the script file in the same directories and in the same order that COMMAND searches for a com-file: the current default directory, then OA$LIB.

SCRIPT-invoked script files may contain two types of entries: text and directives. Directives are identified by a period in the first column. All other lines are considered text.

6.139.4 Function Qualifiers

None.

6.139.5 Examples

In Named Data:

Named Data:

```
1 Name WHAT
SCRIPT ANSWERS
```

Selecting menu option WHAT calls the SCRIPT function, which invokes ANSWERS.SCP.

Interactively:

```
<COMMAND PREPARE\SCRIPT EXECUTE
```

EXECUTE.SCP is invoked after PREPARE.COM finishes.

In a script:

```
!
! Call script CONTXTCHK.SCP
!
.FX SCRIPT CONTXTCHK
```

6.139.6 Special Features and Restrictions

A SCRIPT-invoked script runs in SCRIPT Mode, one of two script Modes. The other mode is DO Mode. In SCRIPT Mode, the script file answers ALL-IN-1 requests for input. In DO Mode, the user answers ALL-IN-1 requests for input—this is typical in ALL-IN-1.

6.140 SET__DEFAULT

6.140.1 Syntax

SET__DEFAULT directory__symbol

6.140.2 Description

The SET__DEFAULT function sets the default directory to the one specified.

You call SET__DEFAULT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.140.3 Function Parameters

directory__symbol

The single, required SET__DEFAULT parameter is an ALL-IN-1 symbol that represents a directory to which the user wants to switch.

6.140.4 Function Qualifiers

None.

6.140.5 Examples

Interactively:

```
<SET__DEFAULT LOG__NAM
```

where LOG__NAM is a system, group, job, or process logical name pointing to a valid VMS directory specification.

6.140.6 Special Features and Restrictions

Calling SET__DEFAULT is equivalent to calling the DCL SET DEFAULT command from the subprocess. SET__DEFAULT does not reassign logical OAUSER, so any file stored in OAUSER and explicitly called as OAUSER:file is found. However, SET__DEFAULT may affect COMMAND, MERGE, SCRIPT and other functions that search for and open files since these functions search the default directory first when you do not provide them with explicit device and directory prefixes. Also, calling SET__DEFAULT may cause errors in single-line DCL commands and on entry forms where the .FILE directive identifies a data file with no device or directory specified.

Any problems in these cases usually arise from the assumption that COMMAND, MERGE, SCRIPT, entry forms, and so on, are always called while OAUSER is the default directory: File not found errors are generated. If you call SET__DEFAULT regularly, you can avoid this sort of error by always specifying a logical device and directory (such as OAUSER or OA$LIB) with file-name parameters in function calls, with command procedures called from within other ALL-IN-1 command procedures (for example, @OA$LIB:FINDCOM), and in .FILE directives on entry forms.

@OA$LIB:FINDCOM.COM provides another way to find command procedures. This command procedure simulates the search strategy used by the COMMAND, DO, and SCRIPT functions.

6.141 SET__MENU

6.141.1 Syntax

SET__MENU [form]

6.141.2 Description

The SET__MENU function specifies a menu as the default for Named Data searches.

You can call SET__MENU from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.141.3 Function Parameters

form

The single, optional SET__MENU parameter is a form that Form Processing uses as the default form when searching for a match to menu input and to keypad definitions. When you omit this parameter, SET__MENU sets default to memory-resident form DEFAULT.

6.141.4 Function Qualifiers

None.

6.141.5 Examples

In Named Data:

Interactively:

```
<SET_MENU NWMAIN
```

This interactive function call causes the Form Processing facility to search the Named Data of form NWMAIN rather than that of form DEFAULT when a user-input match is not found in the Named Data of the current menu.

6.141.6 Special Features and Restrictions

Since some features are available at the user level only through entries in the form DEFAULT's Named Data, the Named Data of any new default form must be constructed with care. For example, keypad key 0 owes its EXIT SCREEN functionality to a OA$FLD_EXIT function call in the Named Data of DEFAULT. Similarly, the exit (EX) option is valid from any menu because it is defined in form DEFAULT's Named Data. Table 4-2 lists the Named Data of DEFAULT.

6.142 SHOW__FORM

6.142.1 Syntax

SHOW__FORM form

6.142.2 Description

SHOW__FORM displays a form but does not process it; thus the form does not have to have any Named Data entries. This function is mainly used to view forms while debugging or demonstrating them.

You can call SHOW__FORM from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.142.3 Function Parameters

form

The single, required SHOW__FORM parameter is the name of the form to display. All that is required is that the form be stored in a form library that is open and accessible to the user. The form remains on the screen until a subsequent function call or user action clears the screen or replaces the form.

6.142.4 Function Qualifiers

None.

6.143 SORT__LIST

6.143.1 Syntax

SORT__LIST input__file__sym,output__file__sym,spec__file__sym
[,spcflag][,pflag][,aflag][,scrflag]

6.143.2 Description

The SORT__LIST function calls WPS-PLUS Sort Processing to sort a list file. When WPS-PLUS is available under ALL-IN-1, you can call SORT__LIST from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts

6.143.3 Function Parameters

SORT__LIST takes three required parameters and four optional ones:

input__file__sym

The first SORT__LIST parameter is an ALL-IN-1 symbol that represents the name of a WPS-PLUS-formatted input file to sort.

output__file__sym

The second parameter is an ALL-IN-1 symbol that represents the name of a file to create, in WPS-PLUS-format, to contain the sorted contents of the input file. The following parameters and flags determine how to create the output file.

spec__file__sym

The third parameter is an ALL-IN-1 symbol that represents the name of a WPS-PLUS-formatted specification file. This file contains instructions on selecting a subset of the elements in the input file.

The following flags determine how SORT__LIST modifies an existing output file.

spcflag

The optional parameter is a supersede flag. When this parameter is 1 and when you have not supplied any other flag parameters, SORT__LIST creates a new output file that supersedes any preceding version of the file. When you omit this parameter or when you supply a value other than 0, SORT__LIST assumes a default of 1. When you supply a spcflag value of 0, SORT__LIST overwrites any existing version of the file, guided by the values, if any, of pflag and aflag.

pflag

The optional parameter is a prefix flag. When this parameter is 1, SORT__LIST inserts its output at the beginning of the specified output file, if it exists. If the file does not exist, SORT__LIST creates it.

When you omit this parameter or when you supply a value other than 1, SORT__LIST assumes a default of 0.

aflag

The optional parameter is an append flag. When this parameter is 1, SORT__LIST appends its output at the end of the specified output file, if it exists. If the file does not exist, SORT__LIST creates it.

NOTE: If more than one of aflag, pflag or spcflag is specified as a 1, then the order of precedence is: appending,prefixing,superseding.

When you omit this parameter or when you supply a value other than 1, SORT__LIST assumes a default of 0.

scrflag

The optional parameter is a screen flag. When this parameter is 1, SORT__LIST displays the developing output on the terminal screen.

When you omit this parameter or when you supply a value other than 1, SORT__LIST assumes a default of 0.

6.143.4 Function Qualifiers

None.

6.143.5 Examples

In a command procedure:

```
$ WRITE OAMAILBOX "OA SORT__LIST #IFILE, #OFILE,#SFILE, -"
$ WRITE OAMAILBOX "OA ''supersede','prefix','append',1"
$ @DCLMAILBOX:
```

In this example, the three required file-name parameters are passed as local symbols, and the optional flags are passed as DCL symbols and a string literal.

6.143.6 Special Features and Restrictions

SORT__LIST is only valid if WPS-PLUS is linked with ALL-IN-1 during installation, as covered in the *Installation Guide.*

6.144 SPAWN

6.144.1 Syntax

SPAWN [command-line]

6.144.2 Description

SPAWN creates a new subprocess with DCL as the Command Language Interpreter (CLI). If you include anything besides the SPAWN command in the function call, ALL-IN-1 treats it as a DCL command to the spawned subprocess. SPAWN deletes the subprocess when the command line has been processed. If you do not specify any parameters, then SPAWN places you in the spawned subprocess in interactive DCL Mode.

You call SPAWN from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.144.3 Function Parameters

[command-line]

You optionally specify a command line after the SPAWN command. SPAWN treats this line as a DCL command to the subprocess. After DCL executes the command, SPAWN deletes the subprocess.

When you omit the command-line parameter, SPAWN places you in the newly spawned subprocess at a DCL $ prompt; you must log out of the subprocess with the DCL LOGOUT command to return to the point of call.

6.144.4 Function Qualifiers

There are no SPAWN qualifiers, but the command line may contain DCL command qualifiers.

6.144.5 Special Features and Restrictions

The spawned subprocess is not the same as the regular ALL-IN-1 subprocess.

6.145 STATISTICS

6.145.1 Syntax

STATISTICS

6.145.2 Description

STATISTICS loads special symbols with VMS process statistics for the main process and displays those statistics on a special form.

You call STATISTICS from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.145.3 Function Parameters

STATISTICS takes no parameters. It updates the STAT$... special symbols with VMS process statistics every time you call the function. STATISTICS automatically displays the values of these symbols through memory-resident argument form OA$STAT.

6.145.4 Function Qualifiers

None.

6.145.5 Examples

In Named Data:

Named Data

```
1     Name ST
STATISTICS
```

Menu option ST calls form OA$STAT to display the current statistics.

Interactively:

```
<STAT
```

6.146 TEST_SPEC

6.146.1 Syntax

TEST_SPEC spec_file_sym

6.146.2 Description

The TEST_SPEC function calls WPS-PLUS List Processing to test a specification file for correctness. When WPS-PLUS is available under ALL-IN-1, you call TEST_SPEC from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts

6.146.3 Function Parameters

spec_file_sym

The single, required, TEST_SPEC parameter is an ALL-IN-1 symbol that represents the name of a WPS-PLUS-formatted list processing specification file. The MERGE_LIST function allows you to specify a specification file when creating a list-processed output document. TEST_SPEC tests that specification file for validity.

6.146.4 Function Qualifiers

None.

6.146.5 Examples

In a command procedure:

```
$ WRITE OAMAILBOX "OA TEST_SPEC #SFILE"
$ @DCLMAILBOX:
```

The required spec-file parameter is passed as a local symbol, #SFILE.

6.146.6 Special Features and Restrictions

TEST_SPEC is only valid if WPS-PLUS is linked with ALL-IN-1 during installation , as covered in the *Installation Guide.*

6.147 TIME

6.147.1 Syntax

TIME

6.147.2 Description

TIME displays the current time on line 24 of the terminal.

You call TIME from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.147.3 Function Parameters

There are no TIME parameters. The following message displays on line 24 of the screen:

```
The current time is hh:mm
```

6.147.4 Function Qualifiers

None.

6.147.5 Examples

In Named Data:

Named Data

```
1 Name TI

TIME
```

Menu option TI displays the current time on line 24.

6.148 UNWIND

6.148.1 Syntax

UNWIND

6.148.2 Description

Whenever you call a menu from a nonmenu, ALL-IN-1 saves the original menu on a prior-menu stack. UNWIND clears this stack.

Form Processing calls this function internally when you call a menu in some nonstandard fashion: for example, from a /POST field qualifier on an entry form.

You call UNWIND from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.148.3 Function Parameters

None.

6.148.4 Function Qualifiers

None.

6.148.5 Examples

Interactively:

```
<UNWIND
```

6.148.6 Special Features and Restrictions

If you call UNWIND from the User Profile-defined starting form, ALL-IN-1 exits.

If you call this function from a menu that you have called from within the editor, that menu exits, and you re-enter the editor. For example, when you press Gold M to display the editor menu, EDTMEN, you begin a new prior-menu stack. You display any number of additional menus here by calling them from the choice field of EDTMEN; EXIT SCREENS will always eventually return you to EDTMEN because keypad 0 (by default) calls OA$FLD_EXIT from those other menus. However, the Named Data of EDTMEN redefines keypad 0 to call UNWIND, allowing you to get back to the editor in a consistent manner.

6.149 VERSION

6.149.1 Syntax

VERSION

6.149.2 Description

The VERSION function displays the current version and base level of ALL-IN-1 on line 24. You call this function from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.149.3 Function Parameters

VERSION takes no parameters. It displays a message containing the different version numbers of the components within ALL-IN-1. For example,

```
A2 BL N.N, MAIL+ BL N.N
```

where BL NN is base level NN. Base levels are a more detailed way to keep track of ALL-IN-1 development within DIGITAL.

6.149.4 Function Qualifiers

None.

6.149.5 Examples

Interactively:

```
<VERSION
```

The current version displays on line 24.

6.150 WAIT

6.150.1 Syntax

WAIT [text]

6.150.2 Description

WAIT creates a pause in ALL-IN-1 processing. This function displays a message wherever the cursor is positioned and waits for a response. Control passes to the user, and WAIT remains active until the user presses a terminator such as RETURN, ENTER, <ESC>, and so on.

You call WAIT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.150.3 Function Parameters

text

You can optionally specify a message of up to 65-characters as the WAIT parameter. When you call WAIT with no text, the message:

```
Press RETURN to continue
```

displays. WAIT displays this message wherever the cursor is currently positioned. ALL-IN-1 truncates the message if it does not fit in the space available.

6.150.4 Function Qualifiers

None.

6.150.5 Examples

In Named Data:

Named Data

```
1 Name S
EDIT 'MSG.TMP'\IFEX\MAIL SEND msg\CLE\WAIT Message sent; press RETURN
```

Menu option S initiates a series of function calls beginning with EDIT. If the user does not quit the edit session, MSG.TMP is sent as a mail message. Then the screen clears, and the following message displays at the top of the screen.

```
Message sent; press RETURN
```

In a command procedure:

```
$ WRITE OAMAILBOX "OA CLEAR"
$ @DCLMAILBOX:
$ TYPE SYS$INPUT
```

A GAMES directory has just been set up for you to hold files created using the GAMES application.

To run GAMES from ALL-IN-1, call Form GMS and select option R.

If you have any problems contact your system manager.

```
$ WRITE OAMAILBOX "OA WAIT"
$ @DCLMAILBOX:
```

This example illustrates the most common use of WAIT. It causes information to remain on the screen for the user to read.

6.150.6 Special Features and Restrictions

Like PROMPT, WAIT is an interactive function that assigns values to the DCL symbols TERMINATOR and RESULT based on user input. Unlike PROMPT, WAIT does not recognize Named Data key definitions, such as keypad 0 (EXIT SCREEN). Although you use WAIT to obtain user input, the PROMPT function should be used instead.

6.151 WPSPLUS$CBI__CLOSE__WINDOW

6.151.1 Syntax

WPSPLUS$CBI__CLOSE__WINDOW

6.151.2 Description

WPSPLUS$CBI__CLOSE__WINDOW is the converse to WPSPLUS$CBI__OPEN__WINDOW. If the script window already exists, it is cleared, and then deleted.

You call WPSPLUS$CBI__CLOSE__WINDOW from the following locations:

- In Named Data
- Interactively
- In scripts

6.151.3 Function Parameters

None.

6.151.4 Function Qualifiers

None.

6.151.5 Examples

From a script:

```
.function WPSPLUS$CBI__CLOSE__WINDOW
```

6.151.6 Special Features and Restrictions

WPSPLUS$CBI__CLOSE__WINDOW requires the optional WPS-PLUS word processing subsystem.

6.152 WPSPLUS$CBI__CLOSE__WINDOW2

6.152.1 Syntax

WPSPLUS$CBI__CLOSE__WINDOW2

6.152.2 Description

WPSPLUS$CBI__CLOSE__WINDOW2 is the converse to WPSPLUS$CBI__OPEN__WINDOW2. If the script window already exists, it is cleared, then deleted.

You call WPSPLUS$CBI__CLOSE__WINDOW2 from the following locations:

- In Named Data
- Interactively
- In scripts

6.152.3 Function Parameters

None.

6.152.4 Function Qualifiers

None.

6.152.5 Examples

From a script:

```
.function WPSPLUS$CBI__CLOSE__WINDOW2
```

6.152.6 Special Features and Restrictions

Requires the optional WPS-PLUS word processing subsystem.

6.153 WPSPLUS$CBI__OPEN__WINDOW

6.153.1 Syntax

```
WPSPLUS$CBI__OPEN__WINDOW line,size
```

6.153.2 Description

WPSPLUS$CBI__OPEN__WINDOW creates a subwindow and clears the window. A CBI script can then write to that window.

WPSPLUS$CBI__CLOSE__WINDOW clears and closes the subwindow.

You call WPSPLUS$CBI__OPEN__WINDOW from the following locations:

- In Named Data
- Interactively
- In scripts

6.153.3 Function Parameters

The two required parameters are line and size. These parameters must be separated by a comma.

Line is the top row where the window is to appear.

Size is the height of the window.

6.153.4 Function Qualifiers

None.

6.153.5 Examples

From a script to open a two line window at row 1:

```
.function WPSPLUS$CBI__OPEN__WINDOW 1,2
```

6.153.6 Special Features and Restrictions

WPSPLUS$CBI__OPEN__WINDOW requires the WPS-PLUS word processing subsystem.

6.154 WPSPLUS$CBI__OPEN__WINDOW2

6.154.1 Syntax

WPSPLUS$CBI__OPEN__WINDOW2 line,size

6.154.2 Description

WPSPLUS$CBI__OPEN__WINDOW2 creates a subwindow and clears the window. A CBI script can then write to that window.

If this function is called and a script window already exists, that window is deleted, and then the new window is created.

WPSPLUS$CBI__CLOSE__WINDOW2 clears and closes the subwindow.

You call WPSPLUS$CBI__OPEN__WINDOW2 from the following locations:

- In Named Data
- Interactively
- In scripts

6.154.3 Function Parameters

The two required parameters are line and size and they must be separated by a comma.

Line is the top row where the window is to appear.

Size is the height of the window

6.154.4 Function Qualifiers

None.

6.154.5 Examples

From a script to open a two-line window at row 1:

```
.function WPSPLUS$CBI__OPEN__WINDOW2 1,2
```

6.154.6 Special Features and Restrictions

Requires the optional WPS-PLUS word processing subsystem.

6.155 WPSPLUS__SPELL

6.155.1 Syntax

WPSPLUS__SPELL file-name

6.155.2 Description

WPSPLUS__SPELL causes the WPS-PLUS editor package to be available as a callable routine. It also sets up to perform SPELL checking on an entire document automatically.

You call WPSPLUS__SPELL from within Named Data.

6.155.3 Function Parameters

File-name

This single required parameter specifies the VMS file-name of the document that is to be checked.

6.155.4 Function Qualifiers

None.

6.155.5 Examples

WPSPLUS__SPELL CAB$:DOCUMENT.FILENAME[@#CURDOC]

Where @#CURDOC is the current document key.

6.155.6 Special Features and Restrictions

Requires the WPS-PLUS document processing subsystem.

6.156 WRITE

6.156.1 Syntax

WRITE mode formname keyfield=symbol [,field=symbol...]

6.156.2 Description

Use WRITE to access data files directly, without an entry-form interface. WRITE is valid for any file that is described by an entry form with a .FILE directive.

Since entry forms describe indexed Record Management Services (RMS) files, you cannot use WRITE to change or add to an RMS sequential file.

You call WRITE from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.156.3 Function Parameters

The first parameter is the mode in which you want to access the data file. The mode may be one of the following:

ADD	— Add a record
CHANGE	— Change an existing record
DELETE	— Delete an existing record

The mode overrides any mode specified by the /MODE entry-form qualifier.

formname

The second parameter is the name of the form that corresponds to the data file you are accessing. The form must have a .FILE entry.

keyfield=symbol...

The third parameter is the field that contains the primary key. This parameter takes a single argument: any ALL-IN-1 symbol that corresponds to the key of the record that you are adding, deleting, copying or modifying.

Subsequent field=symbol parameters are optional, but when you specify them, the symbol contains a new value to be written to the field, not the existing value (if any).

6.156.4 Function Qualifiers

None.

6.156.5 Examples

Interactively:

```
<WRITE ADD FORMAT FORMAT="COMMAND", DESC="Com procedures"
```

6.156.6 Special Features and Restrictions

WRITE is not recommended with File-Cabinet-related files such as OAUSER:DOCDB.DAT. Although this particular file is described by an entry form, the information it contains is part of a larger data set, and altering it with WRITEs may cause unpredictable results in subsequent CABINET function calls.

When you use this function with a file that has segmented keys, the key must be a symbol that contains a concatenation of all the segments, otherwise WRITE looks for a single key. Use the special field name %KEY for the name of this field. For example:

```
WRITE CHANGE AIE %KEY= "T" "TO-DO-LIST" $CAL__TO__DO__NUMBER
```

You can use WRITE with FOR to convert data files after you have renamed one or more fields in the corresponding entry form.

6.157 YESNO__PROMPT

6.157.1 Syntax

YESNO__PROMPT [prompt-symbol]

6.157.2 Description

YESNO__PROMPT displays a question by default and requires that the user enter a yes or no answer as appropriate. Otherwise, it is identical to PROMPT.

You call YESNO__PROMPT from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.157.3 Function Parameters

prompt-symbol

The single, optional YESNO__PROMPT parameter is an ALL-IN-1 symbol that contains the text of the prompt. When omitted, the message `Do you really want to do this [Y/N]?:` displays. Like other ALL-IN-1 symbols, the parameter may comprise a list of different symbol types.

ALL-IN-1 waits until the user responds Y, N, YES, or NO. YESNO__PROMPT is not case-sensitive and recognizes any combination of upper and lowercase letters. The user's intent is stored in OA$FORM__DISPOSE as a 2 for Y or YES and a 0 for N or NO. Also, the actual answer is stored in global DCL symbol RESULT, the ALL-IN-1 symbol OA$PROMPT__TEXT, and the actual terminator-key sequence is stored in special symbol OA$FORM__TERMINATOR.

6.157.4 Function Qualifiers

None.

6.157.5 Examples

Interactively:

```
<YESNO_PROMPT
```

Calling YESNO_PROMPT with no parameters from a menu displays the following message on line 24:

```
Do you really want to do this [Y/N]?:
```

ALL-IN-1 suspends processing until the user enters an appropriate answer and presses RETURN or EXIT SCREEN.

```
<YESNO_PROMPT "Y or N"
```

The quoted string displays. YESNO_PROMPT expects the same response as with the default prompt string.

In a command procedure:

```
$ WRITE OAMAILBOX "OA YESNO_PROMPT\GET OA$FORM_DISPOSE"
$ @DCLMAILBOX:
$ IF "'RESULT'" .EQS. "0" THEN EXIT
$!
$! Continue processing...
$!
```

The YESNO_PROMPT call determines whether the user wants to continue with the procedure. It is easier to test the value of OA$FORM_DISPOSE rather than the actual characters entered, so GET OA$FORM_DISPOSE is called in the same WRITE to OAMAILBOX. This causes the disposition of the YESNO_PROMPT call (that is, the value of OA$FORM_DISPOSE), rather than the actual user input, to return to symbol RESULT.

6.157.6 Special Features and Restrictions

When a user answers no to a YESNO__PROMPT, the Form disposition is set to EXIT (the same as OA$FLD__EXIT). This may cause a form to exit. To prevent a premature exit, follow the YESNO__PROMPT function call with a call to the OA$FLD__STAY function. For example:

```
.NAME D
YESNO__PROMPT\OA$FLD__STAY\.IF OA$PROMPT__TEXT = 'Y' THEN
  CABINET__DELETE__DOCUMENT
```

See also the PROMPT function in this chapter.

6.158 YESNO__PROMPT__ID

6.158.1 Syntax

YESNO__PROMPT__ID prompt-id [prompt-symbol]

6.158.2 Description

YESNO__PROMPT__ID is the same as YESNO__PROMPT, except that the first word after the function name is saved, for the duration of the prompt, as the prompt's name. The prompt name is stored in the ALL-IN-1 symbol OA$PROMPT__ID and is not displayed as part of the prompt.

If a SCRIPT Mode script is active when the script is invoked, the script can test OA$PROMPT__ID to identify the prompt.

See Chapter 8 for more information about OA$PROMPT.

YESNO__PROMPT__ID displays a question by default and requires that the user enter a yes or no answer as appropriate. Otherwise, it is identical to PROMPT.

You call YESNO__PROMPT__ID from the following locations:

- In Named Data
- Interactively
- Through OAMAILBOX from the subprocess
- In templates
- In scripts
- Within validation expressions

6.158.3 Function Parameters

prompt-id

This required parameter specifies the prompt name that is stored in OA$PROMPT.

prompt-symbol

The optional YESNO__PROMPT__ID parameter is an ALL-IN-1 symbol that contains the text of the prompt. When omitted, the message `Do you really want to do this [Y/N]?:` displays. Like other ALL-IN-1 symbols, the parameter may comprise a list of different symbol types.

ALL-IN-1 waits until the user responds Y, N, YES, or NO. YESNO__PROMPT__ID is not case-sensitive and recognizes any combination of upper and lowercase letters. The user's intent is stored in OA$FORM__DISPOSE as a 2 for Y or YES and a 0 for N or NO. Also, the actual answer is stored in global DCL symbol RESULT, and the actual terminator-key sequence is stored in special symbol OA$FORM__TERMINATOR.

6.158.4 Function Qualifiers

None.

6.158.5 Special Features and Restrictions

See also the PROMPT function in this chapter.

Index

This index references pages in all volumes: Volume 1 comprises the Preface through Chapter 5; Volume 2 comprises Chapter 6; Volume 3 comprises Chapter 7 through the Appendixes and the Glossary.

" and '
  string delimiters
    and /COMPUTE, **4-7**
    and /GET form qualifier, **3-16**
    and /HARD, **3-16, 4-18**
    and /LOG, **3-19**
    and /POST_FUNCTION, **3-21, 4-24**
    and /PRE_FUNCTION, **3-21, 4-25**
    and /PROMPT, **4-26**
    and /SHOW, **4-45**
    and COMMAND function parameters, **6-32**
    and CXP SEND, **12-8**
    and CXP WAIT, **12-9**
    and FOR function RSEs, **6-88**
    and GET, **6-107**
    and OA$TABLE, **C-8**
    and QUEUE_BATCH qualifiers, **6-280**
    and QUEUE_PRINT qualifiers, **6-285**
    in symbols, **8-2**
    in subprocess *See also* ', **7-6**
",'
  string delimiters
    in DSRs, **8-6**
#
  local-symbol prefix, **8-4**
$
  and DCL function, **6-51**
  permanent-symbol prefix, **8-3**

&
MERGE directive prefix, 6-138
'
See also " and '
symbol-substitution character
in subprocess, 7-6
', " and string delimiters
and OA$TABLE, C-8
-
continuation character
and COMPUTE, 6-41
in WRITEs
to OAMAILBOX, 7-7
.
as field identifier in
FOR-function RSEs, 6-89
FORM-function substitution
character, 6-95
substitution character, 3-4
/
function-qualifier delimiter, 6-3
:
as a symbol-substring delimiter,
8-10
as an alternate-key delimiter
in DSRs, 8-6
Mail distribution list suffix,
11-3
<
function-call prefix, 3-32, 6-2
in a validation expression,
4-20, 4-30, 4-49, 4-54
pipeline source prefix, 6-9
<->
continuation characters
for MERGE template
directives, 6-139
=
buffer identifier in default
editor, 10-5
Help cross-reference prefix,
14-16
@
iterative symbol substitution
prefix, 8-12
Mail distribution list prefix,
11-3
Remote-node mail prefix, 11-4
\
function-call delimiter
function list, 3-21
in subprocess, 7-7
in user-defined
functions, 6-11
\\
function call delimiter
in .IF expressions, 15-18
function-call delimiter
in FOR-function calls, 6-87
_
SPECIAL mail destination
prefix, 11-4

A
ACMSTASK function, 6-13 to
6-15
Action Items
checking
with AI_CHECK, 6-17 to
6-18
class, 3-42
entry form AIE
field information, 3-40
item, 3-42
priority, 3-42
schedule date, 6-17
status, 6-17
storage of, 3-40
type, 3-42, 6-17
Additional options
and Gold A, 4-56
ADE function, 6-16
AI_CHECK function, 6-17 to
6-18
ALL-IN-1
and the VAX Information
Architecture, 1-4
customizing
suggested prerequisites, xiii
exercises, A-1 to A-22

facilities
primary, 1-7
flow control, 1-4
organization, 1-6
overview, 1-1 to 1-7
performance
See Performance
subsystems
overview, 1-6
symbols, 8-1 to 8-36
APPEND function, 6-19 to 6-20
Application
debugging an
with Gold (, 6-34
defined, 1-6
design guidelines, 16-18
"standard", 16-19
subsystem versus, 1-6
APR
conventions, 7
organization of, 2
overview, 1 to 8
related documents, 4, 5 to 7
using the, 3
Architecture, VAX Information
overview, 1-3
Argument form, 3-56 to 3-57
Named Data, 3-56
qualifiers, 3-57
.TYPE ARG, 3-5
ATTACH function, 6-21 to 6-22
Attributes, document, 15
Attributes, FMS field
used to define a field, 4-2
/AUTO field qualifier, 4-5
and /RECOGNITION, 4-28
and /RSE_RECOG, 4-31
Autorecognition
with /AUTO, 4-5

B

BACKSPACE
and OA$FLD_PREV, 4-56
Batch jobs
See QUEUE_BATCH
/BEGIN form qualifier, 3-10, 3-27, 3-47
and /FIELDS, 3-49
in menus, 3-11
/BLANK field qualifier, 4-5 to 4-6
Boolean Expression, 4
Boolean expression, 6-86
in CABINET SCAN calls, 9-44
in FOR-function calls, 6-85
in search-form RSE, 3-60, 3-63
Boolean operators, 4
in .SFILE directive, 3-66
in FOR-function calls, 6-85
in search form RSE, 3-64
Brace character, left
used as script mnemonic
delimiter, 15-3
Brace character, right
used as script mnemonic
delimiter, 15-3
BYE function, 6-23 to 6-24

C

CAB$
File Cabinet DSR, 9-16
CAB$ATTACH, 9-17
CAB$ATTRIBUTES, 9-17
in a search form, 3-63
in a select form, 3-71
CABINET function, 6-25, 9-20 to 9-47
subfunctions
See also CABINET HELP
ADD_ATTRIBUTE, 9-21
ATTACH_DOCUMENT, 9-21
BEGIN, 9-22
CHANGE_ATTRIBUTE, 9-22
CLEAR_CURDOC, 9-23
CLOSE, 9-23
CONVERT, 9-23
COPY, 9-24
CREATE, 9-25

CABINET function,
CROSSFILE, 9-26
CURRENT, 9-27
DECREMENT_COUNT, 9-27
DELETE_ATTRIBUTE, 9-27
DELETE_DOCUMENT, 9-28
DELETE_OR_REFILE, 9-28
DETACH_DOCUMENT, 9-29
EDIT_DOCUMENT, 9-30
END, 9-30
EXCHANGE, 9-30
FILE_ATTACHMENT, 9-30
FORCE, 9-31
GET_PENDING, 9-31
HELP, 9-32
JANITOR, 9-33
NEXT_DOCUMENT, 9-33
NEXT_FOLDER, 9-35
NEXT_NUMBER, 9-37
NEXT_SCAN, 9-38
NEXT_TITLE, 9-40
OPEN, 9-42
POP, 9-42
PUSH, 9-42
PUT_PENDING, 9-42
REFILE_DOCUMENT, 9-43
REORG_DOCDB, 9-44
RESET_DELETE, 9-44
See also CABINET RESET_DELETE
SCAN, 9-44
SELECT, 9-45
SHARE_DOCUMENT, 9-9, 9-46
STOP_SCAN, 9-47
summary, 9-13
UNSHARE_DOCUMENT, 9-47
Calculation form, 3-57 to 3-59
behavior, 3-57
Named Data, 3-57
qualifiers, 3-58
.TYPE CALC, 3-6
.TYPE CLEAR, 3-6
.TYPE DTR, 3-6
CALENDAR function, 6-26
current date/time, 13-2
data sets, 13-1
used by, 13-30
files used by, 13-3
subfunctions, 13-3 to 13-27
CALENDAR ADD_ATTENDEE, 13-3
CALENDAR CANCEL APPOINTMENT, 13-4
CALENDAR CANCEL MEETING, 13-4
CALENDAR CLEAR ATTENDEES, 13-5
CALENDAR CONVERT ACTION_ITEMS, 13-5
CALENDAR CONVERT APPOINTMENTS, 13-5
CALENDAR CONVERT MEETINGS, 13-6
CALENDAR DISPLAY ANSWER, 13-6
CALENDAR DISPLAY APPOINTMENT, 13-7
CALENDAR DISPLAY GRAPH, 13-8
CALENDAR DISPLAY MEETING, 13-9
CALENDAR DISPLAY MONTH, 13-9
CALENDAR DISPLAY NEXT APPOINTMENT, 13-10
CALENDAR DISPLAY NEXT MEETING, 13-11
CALENDAR DISPLAY NEXT REPLY, 13-11
CALENDAR DISPLAY REPLY, 13-12
CALENDAR INITIALIZE FILES, 13-13
CALENDAR INITIALIZE MONTH, 13-13

CALENDAR MAIL_TO
_ATTENDEES, 13-14
CALENDAR
MAKE_ATTENDEE_
SYMBOLS, 13-14
CALENDAR
MAKE_MEETING_
SYMBOLS, 13-15
CALENDAR PRINT, 13-16
CALENDAR PRINT
ALL, 13-16
CALENDAR PRINT
CUSTOM, 13-17
CALENDAR PRINT
DAY_REMIND, 13-17
CALENDAR PRINT
DAY_SCHEDULE, 13-18
CALENDAR PRINT
MONTH, 13-18
CALENDAR PRINT
TODO, 13-19
CALENDAR PRINT
TWOCAL, 13-19
CALENDAR PRINT
WEEK_REMINDERS,
13-20
CALENDAR PRINT
WEEK_SCHEDULE,
13-20
CALENDAR PURGE, 13-20.1
CALENDAR
REMOVE_ATTENDEES,
13-20.1
CALENDAR
REPLY_TO_MEETING,
13-21
CALENDAR RESCHEDULE
APPOINTMENT, 13-21
CALENDAR RESCHEDULE
MEETING, 13-22
CALENDAR SCAN, 13-22
CALENDAR SCHEDULE
APPOINTMENT, 13-23
CALENDAR SCHEDULE
MEETING, 13-23
CALENDAR SET
ALTERNATE_USER,
13-24
CALENDAR SET
DATE, 13-25
CALENDAR SET
NETWORK OFF, 13-26
CALENDAR SET
NETWORK ON, 13-26
CALENDAR VERSION,
13-27
/CAPTIVE menu qualifier, 3-26
CBI
exit keys
Gold X, CTRL/C, 4-59, 15-4
Character, left Brace
used as script mnemonic
delimiter, 15-3
Character, right Brace
used as script mnemonic
delimiter, 15-3
CHECK_DISKQUOTA function,
6-26.1 to 6-26.2
/CHOICE menu qualifier, 3-25
/CLEAR field qualifier, 4-6
/CLEAR form qualifier, 3-11
and /CHOICE, 3-11
CLEAR function, 3-21, 6-27 to
6-28
versus /OVERLAY, 6-28
CLOSE_PRIOR function, 6-29 to
6-30
in function lists, 6-7
COMMAND function, 6-31 to 6-36,
7-19
and mailboxes, 7-2
debugging tools used with, 6-34
passing parameters with, 7-16
simulating search strategy of
with OA$LIB:FINDCOM,
6-35
versus QUEUE_BATCH, 6-282
Command procedure
debugging
with Gold (, OA$LIB:V.COM,
6-34

Command procedures versus
scripts performance in
applications, 16-22
Common Data Dictionary (CDD), 5
Communications Control, 12-2 to
12-13
See also CXP function
and the CXP function, 6-47
Control Documents, 12-2
customizing, 12-12
KEYPAD function, 6-117
Terminal Emulation Mode, 12-2
/COMPUTE field qualifier, 4-7
COMPUTE function, 6-37 to 6-41
from the subprocess, 7-10, 7-12
Compute key
Gold #, 4-58
Control Documents, 12-2
CONTROL_C function, 6-42
Converting to Version 2.X
form libraries, 16-9
/COPY field qualifier, 3-79, 4-8 to
4-10
order of processing, 4-10
COPY function, 6-43 to 6-44
as a pipeline source, 6-10
in Pipeline Mode, 6-44
Copy Mode, hard, B-1 to B-5
See also MODE function
providing information while in,
4-18
CREATE function, 3-38, 6-45 to
6-46
Customizing ALL-IN-1
performance considerations
See Performance
with user-defined functions,
6-10 to 6-11
Customizing documentation
for HELP, 16-24
for the user, 16-23
Customizing forms, 16-8
CXP function, 6-47, 12-2 to 12-13
See also External
Communications,
Communications Control
subsystem
subfunctions, 12-5
AT, 12-5
CONNECT, 12-5
CONNECT/DEVICE, 12-6
DISCONNECT, 12-6
DISPLAY, 12-6
PAUSE, 12-6
PROCESS, 12-7
RECORD, 12-7
restrictions on calling, 12-9
SEND, 12-8
SEND/DOCUMENT, 12-8
SET, 12-9
STOP RECORDING, 12-9
WAIT, 12-9

D

/DATA entry form qualifier
and /KEY, 3-48
Data file keys
ALL-IN-1 versus RMS, 3-39
specified with FDL, 3-39
Data files
accessing
through alternate keys, 8-6
through DSRs, 8-4 to 8-9
with .FILE, 3-44
with /DATA, 3-44
adding unique key values to
with /INVALID, 4-21
converting
with FOR and WRITE, 6-91
copying
with COPY, 6-43 to 6-44
creating, 3-38, 16-4
with CREATE, 6-45 to 6-46
with OA$INI_LOGICALS,
6-204
deleting, 6-56 to 6-57
modifying
with WRITE, 6-328 to 6-329

updating existing records
with /VALID, 4-49
upgrading
with OA$CNV_CONVERT, 6-161 to 6-162
with OA$CNV_MERGE, 6-165 to 6-166
Data set, 11
See also DSR
text
See Text data sets
Data sets, C-1 to C-15
caching, 6-69, 6-169
CALENDAR, 13-1
data, C-1
See also DSR
form, C-2
in a validation expression, 4-20, 4-29, 4-48, 4-51
in FOR function RSEs, 6-84
permanently open, 6-70
special
See DSR
text, C-2
Data validation
See Validation
See also DTR function
and DTR function, 6-66
and DTR$ special symbols, 8-3
creating record definitions, 16-5
getting information from, 6-66
/DATE form qualifier, 3-12
Date/time formats, 6-49, 8-13
DATE_CONVERT function, 6-48 to 6-50
from the subprocess, 7-10
Dates and times
formatting with DATE_CONVERT, 6-48 to 6-50
DCL
noninteractive commands
with OA$DCL, 6-52
DCL form
.TYPE DCL, 3-6
DCL function, 6-51 to 6-53, 7-19, 16-15
and Gold $, 4-58
and mailboxes, 7-2
called from the editor, 16-16
from the default editor, 10-17
DCL key
Gold $, 4-58
DCL symbols
common, 7-12 to 7-15
and function lists, 7-15
RESULT, 7-12
TERMINATOR, 7-13
creating
with /COPY, 4-8, 7-9
with /PUT_SAVE, 4-27, 7-9
with COMPUTE, 6-38, 7-10
with DATE_CONVERT, 6-48, 7-10
with functions, 7-10 to 7-11
with GET, 6-102, 6-103, 7-11
with MAKE_FILE_NAME, 6-130, 7-11
with NEXT_LIST, 6-155
with NEXT_LIST, OA$FLD_NEXT_LIST, 7-11
with PARSE_USER, 7-11
with SAVE, 6-293, 7-11
verifying assignment of, 16-15
DCLMAILBOX
and OAMAILBOX, 7-4
mailbox to subprocess, 7-4
saving data that comes through
with RESULT, 7-12
with TERMINATOR, 7-13
Debugging
applications
with Gold ((SET VERIFY), 6-34
command procedures
with Gold (, OA$LIB:V.COM, 6-34
scripts
with Gold ((.TRACE ON), 6-34

Debugging applications
with Gold (, 4-62
Debugging features, 16-15
/DECIMAL field qualifier, 4-10 to 4-11
versus DECIMAL function, 4-11
DECIMAL function, 4-11, 6-54 to 6-55
Default menu
form DEFAULT, 3-32, 4-56
keypad definitions in, 3-32
setting
with SET_MENU, 6-300 to 6-301
Defining/redefining the keyboard, 4-63 to 4-65
DELETE_FILE function, 6-56 to 6-57
Desk Calculator form, 3-58 to 3-59, 4-60
formula mode, 3-59
mathematical functions, 3-59
setting decimal places, 3-59
.TYPE DESK, 3-6
using memory registers, 3-59
DESTINATION function, 6-58, 10-10
and the Editor menu, 6-75
DIGITAL Command Language
See DCL
Directives, keyboard
recognized at prompts, 6-269
/DISPLAY form qualifier, 3-12
DISPLAY function, 6-59 to 6-62
Displaying information
through a scrolled region, 4-33
with /SHOW, 4-45
Dispositions, form
terminator values, 4-69
DO function, 6-63 to 6-64
See also SCRIPT
DO key
RETURN or Gold F, 4-58
/DOC field qualifier, 4-11 to 4-14
and MAKE_FILE_NAME, 6-133
Document attributes, 15
Document, File Cabinet
See also File Cabinet, Text files
attachments, 9-3, 9-21
attributes, 9-3, 9-8, 9-9, 9-13
See also PDAF, SDAF
adding, 9-21
changing, 9-22
deleting, 9-27
compound, 9-3
detaching from, 9-29
filing attachments, 9-30
copying, 9-24
See also Text files, copying
creating, 9-25
cross-filing, 9-26
current, 9-27
See also OA$CURDOC
deleting, 9-28, 9-33
See also File Cabinet Wastebasket and Text files, deleting
with File Cabinet janitor, 9-4
editing, 9-30
See also Text files, editing
next
by folder, 9-35
by number, 9-37
by title, 9-40
in a scan, 9-38
in folder, 9-33
number, 9-2
ordering, 9-6
refiling, 9-43
removing from shared area, 9-47
selecting, 9-45
sharing, 9-3, 9-46
Documentation, customizing
for HELP, 16-24
for the user, 16-23

Down Arrow
and OA$FLD_DOWN, 4-58
calls OA$SCL_DOWN in scrolled region, 4-34
DSR, 8-4 to 8-9
alternate keys with, 8-6
and /DATA, 3-48
combining and nesting, 8-6
File Cabinet, 9-16
See also CAB$
in a validation expression, 4-51
with /INVALID, 4-20
with /RECOGNITION, 4-29
with /VALID, 4-48
obtaining data with, 8-8
scrolling
SCROLL$FUNCTION, 4-36 to 4-42
special, C-2
OA$DIR, C-3
special field names
%ADDRESS, 8-7
%COUNT, 8-7
%KEY, 8-8
%NEXT, 8-8
%WHOLE, 8-8
syntax, 8-4
validation
OA$DATE, C-2
OA$TABLE, C-8
OA$YESNO, C-9
OA$YN, C-9
DTR function, 6-65 to 6-68
DUMP_CACHE function, 6-69 to 6-70, 16-11
and OA$FLO_CLOSE_LIB, 6-194
and OA$FLO_OPEN_LIB, 6-196

E
Edit form
.TYPE EDIT, 3-6
EDIT function, 6-71 to 6-73
Editing (default editor), 10-4 to 10-19
See also EDIT function
Change Mode, 10-4
commands
nokeypad commands
switching to Line Mode from, 10-18
creating new files from editor, 10-17
edit buffers, 10-5
including text into document, 10-16 to 10-17
initialization file, 10-8
initializing, 10-4
key numbers, 10-12
Line Mode, 10-4
commands, 10-4
APPENDSR, 10-15
BUFFER, 10-5
CHANGE, 10-8, 10-9, 10-19
CLEAR, 10-9
COPY, 10-15
CUTSR, 10-14
DEFINE KEY, 10-6
DEFINE MACRO, 10-5
FIND, 10-9
INCLUDE, 10-16
INSERT, 10-9
PASTE, 10-14
RESET, 10-14
SEL, 10-14
SHOW KEY, 10-11
WRITE, 10-17
macros, 10-5
invoking with key sequences, 10-8
moving text within a document, 10-14 to 10-16
moving within a document, 10-13
nokeypad commands, 10-5
ADV, 10-13
and DEFINE KEY, 10-6
BACK, 10-13

Editing (default editor)
nokeypad commands (Cont.)
BR, 10-14
C, 10-13
DEFK, 10-6
ER, 10-13
EXIT, 10-19
EXTEND (EXT), 10-8
L, EL, 10-13, 10-14
V, 10-13, 10-14
W, 10-13
XLATE, 10-10
and DESTINATION, 10-10
Editing (WPS-PLUS)
converting files, 10-21
Editor menu
and EDT function, 6-74 to 6-75
and function-call output, 6-75
EDT function, 6-74 to 6-75
Electronic Messaging, 11-1 to 11-50
See also MAIL function
alternative MTS, 11-5
current message, 11-1
delivery mechanism, 11-3
distribution lists, 11-3
SUBSCRIBERS:, 11-4
header and text, 11-2
logicals
OA$MAIL_NUM_ADDRESSES, 11-9
mail destinations, 11-4
menu interface, 11-2
nicknames, 11-3
special addressees
and SPECIAL.CMU, 11-6
and SPECIAL.COM, 11-6
@dest @node, 11-4
dist_list:, 11-4
I or ME, 11-3
PAPER MAIL, 11-3
_special-addressee, 11-4
special folders, 11-2
unrecognized destinations, 11-6
Entry form, 3-37 to 3-56
and data sets, 4-51
dynamically redefining, 3-78
key field recognition
and /SHOW, 4-46
Named Data, 3-37
processing, 3-54 to 3-56
required features, 3-37
.TYPE ENTRY, 3-6
using /STORE with, 4-46
ERROR function, 6-76 to 6-77
in function lists, 6-6
Errors
logging
with LOG, 6-124 to 6-125
trapping in function lists with ERROR, 6-6, 6-76 to 6-77
EXIT function, 6-78 to 6-79
EXIT SCREEN
and the /POST_FUNCTION field qualifier, 4-24
and the prior form, 3-75
from the editor menu, 6-315
keypad 0
and OA$FLD_EXIT, 4-59
trapping with IFEXIT, 6-6
EXIT_SCREEN, 3-20
Expression, boolean
in search-form RSE, 3-63
Expression, record selection
See RSE
External Communications, 12-1 to 12-13
See also CXP function, Multinode features, Communications Control subsystem

F

Facility
defined, 1-6
FDL file specification
in entry form Named Data, 3-39
FHELP function, 6-80

Field
as a symbol, 8-2
assigning a permanent value to
with /BLANK, 4-5
assigning values dynamically
with PUT_FIELD, 6-277 to 6-278
attributes
assigned with FMS, 4-2
FMS versus ALL-IN-1, 4-3
clearing
with /CLEAR, 4-6
defined, 4-1
defining
with field qualifiers, 4-2
with FMS field attributes, 4-2
display-only, 3-13
getting values from
with /PUT_SAVE, 4-27
in a DSR, 8-4
layout
viewing with Gold V, 4-63
moving from one to another, 4-66
postprocessing from
with /POST_FUNCTION, 4-24
preparing for mathematical output
with /DECIMAL, 4-10
preprocessing from
with /PRE_FUNCTION, 4-25
prompting through
with /PROMPT, 4-26
putting file names into
with /DOC, 4-11
putting values into
with /COPY, 4-8
with /GET_SAVE, 4-16
suppressing input
with /BLANK, 4-5
validating contents of
with /FILE_CHECK, 4-14
validation, 4-51 to 4-56
See also Validation
ensuring a unique value
with /INVALID, 4-19
/VALID, 4-47
viewing valid values for
with /RECOGNITION, 4-28
FIELD function, 4-2, 6-81 to 6-82
and /FIELDS form qualifier, 4-2, 6-81
and OA$FLD_nnn functions, 4-67 Field processing, 4-1 to 4-70
summary, 4-69 to 4-70
Field qualifiers
/AUTO, 4-5
/BLANK, 4-5 to 4-6
/CLEAR, 4-6
/COMPUTE, 4-7
/COPY, 3-79, 4-8 to 4-10
/DECIMAL, 4-10 to 4-11
/DOC, 4-11 to 4-14
/FILE_CHECK, 4-14 to 4-16
/GET_SAVE, 4-16 to 4-18
/HARD, 4-18 to 4-19
/INVALID, 4-19 to 4-22
/LIST, 4-22 to 4-23
/OPTIONAL, 4-23 to 4-24
/POST_FUNCTION, 4-24 to 4-25
/PRE_FUNCTION, 4-25 to 4-26
/PROMPT, 4-26 to 4-27
/PUT_SAVE, 4-27 to 4-28
/RECOGNITION, 4-28 to 4-30
/RSE_INVALID, 4-30 to 4-32
/RSE_RECOG, 4-30 to 4-32
/RSE_VALID, 4-30 to 4-32
/SCROLL, 4-33 to 4-44
/SHOW, 4-45 to 4-46
/STORE, 4-46
types of
in-field, 4-4
postprocessing, 4-4
preprocessing, 4-4
validation, 4-4

Field qualifiers (Cont.)
/UNIQUE, 4-47
/USE_FORM, 4-47
used to define a field, 4-2
/VALID, 4-47 to 4-50
/FIELDS form qualifier, 3-13, 3-34
and /BEGIN, 3-13, 3-49
and /GET, 3-13
and /KEY, 3-49
and FIELD function, 6-81
in FORM function calls, 3-9
in menus, 3-26
File Cabinet, 9-1 to 9-47
accessing data, 9-15 to 9-18
See also Special symbols and CAB$ data set
as a data set, 9-15
attachments, 9-3
See also Document attachments
converting, 9-23
Document, 9-2
See also Document, Text files attached
See Document attachments compound
See Document, compound
number, 9-2
sharing, 9-3
folder, 9-3
janitor, 9-4
organization, 9-4 to 9-8, 9-13
See also individual file names
personal area, 9-4
reorganizing, 9-44
searching the
with search forms, 3-63
with select forms, 3-71
shared area, 9-19
switching
with NEWDIR, 6-150
using the, 9-13 to 9-19
Wastebasket
and CABINET JANITOR, 9-33
File Cabinet special symbols
See Special symbols, 9-18
File Definition Language
See FDL
.FILE directive
in entry form Named Data, 3-38
File names
creating
with /DOC, 4-11
/FILE_CHECK field qualifier, 4-4, 4-14 to 4-16
Files
See Text files, Data files
copying with COPY, 6-43 to 6-44
deleting
with DELETE_FILE, 6-56 to 6-57
managing
with the File Cabinet, 9-1 to 9-47
naming
with MAKE_FILE_NAME, 6-130 to 6-133
processing, 10-32
purging
with PURGE_FILE, 6-273 to 6-276
Flow control, 2-1 to 2-6
ALL-IN-1, 1-4
between ALL-IN-1 and its subprocess, 7-3
framework, 2-4 to 2-6
FMS field attributes
used to define a field, 4-2
FMS field highlights
performance in applications, 16-23
FMS forms
ALL-IN-1 use of, 3-1
FOR function, 6-83 to 6-91
and /RSE_nnn field qualifiers, 4-32
and validation qualifiers, 6-90

and WRITE
converting files, 6-91
called, 4-31
data sets in, 6-84
DO subfunctions
in select forms, 3-72
in search forms, 6-90
in select forms, 3-69, 6-90
RSEs in, 6-83
selecting unique records
with, 6-84
FORCE function, 6-92 to 6-93
in function lists, 6-7, 6-92
Form
caching, 6-69, 6-169
called directly from menu
choice field, 3-31
creating, 16-4
customizing, 16-8
displaying
with SHOW_FORM, 6-302
dispositions
and OA$FORM_DISPOSE,
4-66
DONE, 4-68
EXIT, 4-68
STAY, 4-68
summary of, 4-68
dynamically redefining, 3-8
maintaining, 16-2 to 16-12
memory-resident, 16-10
modifying, 16-5
multipage
See Form set
nonstandard, 3-80
redefining
with FORM function calls,
3-75 to 3-78
saving image of
with PRINT_SCREEN,
6-265
segmented
See Form set
testing, 16-11
Form (Cont.)
.TYPE
ARG, 3-5
CALC, 3-6
CLEAR, 3-6
DCL (special), 3-6
DESK (special), 3-6
DTR, 3-6
EDIT (special), 3-6
ENTRY, 3-6
HELP (special), 3-6
in FORM function calls, 3-76
LIST (special), 3-6
MENU, 3-6
Named Data entry, 3-5
SEARCH, 3-6
SELECT, 3-6
VIEW, 3-6
Form and function
in ALL-IN-1, 1-5
in the VAX Information
Architecture, 1-3
Form dispositions
terminator values, 4-69
FORM function, 3-3, 6-94 to 6-96
and OA$FLD_nnn
functions, 4-67
form .TYPE entry in calls to,
3-76
in function lists, 6-7
redefining Named Data with,
3-75
substitution character with, 3-4
Form Libraries, 3-2, 16-9
closing
with
OA$FLO_CLOSE_LIB,
6-193 to 6-194
interactively updating, 16-17
opening
with
OA$FLO_OPEN_LIB,
6-195 to 6-196
opening/reopening
with NEWLIB, 6-152 to
6-154

Form Libraries (Cont.)
search order, 3-3
searching, 16-9
status of
and
OA$FBT_LIST_LIBTREE, 6-175
and OA$FBT_LIST_TREE, 6-177
Form processing, 3-1 to 3-80
See also individual form
.TYPES
Form qualifiers, 3-8
in function calls, 3-8
common, 3-10 to 3-24
/BEGIN, 3-10
/CLEAR, 3-11
invalid with entry forms, 3-11
/DATE, 3-12
/DISPLAY, 3-12
/FIELDS, 3-13
/GET, 3-14
/HARD, 3-16
/HELP, 3-17
/HIGHLIGHT, 3-17
/KEYPAD, 3-18
/LOG, 3-19
/MAIL, 3-19
/NEXT, 3-19
/OVERLAY, 3-20
/POST_FUNCTION, 3-21
/PRE_FUNCTION, 3-21
/RESET, 3-22
/SUPER, 3-23
/TITLE, 3-23
/USER, 3-24
entry form
/BEGIN, 3-47
/DATA, 3-48
/FIELDS, 3-49
/KEY, 3-50
/MODE (required), 3-44
/NOWRITE, 3-52
/ONCE, 3-52
/ONE_ENTRY, 3-53
Form qualifiers (Cont.)
/SAVE_START, 3-46, 3-53
in function calls
rules for processing, 3-8
menu
/CAPTIVE, 3-26
/CHOICE (required), 3-25
/FIELDS, 3-26
/MORE, 3-27
/ONCE, 3-28
select form
/LIST, 3-69
/STYLE, 3-70
Form set, 3-78 to 3-80, 20
duplication of field names in, 3-79
entering first field in
with Gold T, 4-62
entering last field in
with Gold B, 4-57
entering next page in
with Gold TAB, NEXT SCREEN, 3-79, 4-60
entering previous page in
with Gold BACKSPACE, PREV SCREEN, 4-61
Gold key functions for, 3-80
restrictions on creating, 3-79
.TYPE directive in, 3-78
.FORM_SET directive
See also Form set
used to create a form set, 3-78
FORMAT entry form, 10-29
FORMAT function, 6-97 to 6-100
as a pipeline source, 6-10
Formatting files
See Text files, formatting
Forms, customizing, 16-8
Forms, precompiling
with the PFC option, 16-11
Function lists
creating loops in
with REPEAT, 6-7
ERROR in, 6-6
forcing displays in
with FORCE, 6-7

from the subprocess, 6-8, 7-7
and common DCL symbols, 7-15
IFEXIT in, 6-6
trapping error conditions in, 6-6
with /POST_FUNCTION qualifiers, 3-21, 4-24
with /PRE_FUNCTION qualifiers, 3-22, 4-25
Functions
See also individual functions
ACMSTASK, 6-13 to 6-15
ADE, 6-16
AI_CHECK, 6-17 to 6-18
APPEND, 6-19 to 6-20
ATTACH, 6-21 to 6-22
BYE, 6-23 to 6-24
called from validation expression, 4-20, 4-30, 4-49, 4-54
called through keyboard directives, 4-63
calling, 6-1
CHECK_DISKQUOTA, 6-26.1 to 6-26.2
CLEAR, 6-27 to 6-28
CLOSE_PRIOR, 6-29 to 6-30
COMMAND, 6-31 to 6-36
COMPUTE, 6-37 to 6-41
CONTROL_C, 6-42
COPY, 6-43 to 6-44
CREATE, 6-45 to 6-46
CXP, 6-47
DATE_CONVERT, 6-48 to 6-50
DCL, 6-51 to 6-53
DECIMAL, 6-54 to 6-55
DELETE_FILE, 6-56 to 6-57
DESTINATION, 6-58
DISPLAY, 6-59 to 6-62
DO, 6-63 to 6-64
DTR, 6-65 to 6-68
DUMP_CACHE, 6-69 to 6-70
EDIT, 6-71 to 6-73
Functions (Cont.)
EDT, 6-74 to 6-75
ERROR, 6-76 to 6-77
EXIT, 6-78 to 6-79
FHELP, 6-80
FIELD, 6-81 to 6-82
FOR, 6-83 to 6-91
FORCE, 6-92 to 6-93
FORM, 6-94 to 6-96
FORMAT, 6-97 to 6-100
general syntax, 6-2
GET, 6-101 to 6-107
HELP, 6-108 to 6-111
IFEXIT, 6-112 to 6-113
INCLUDE, 6-114 to 6-115
interactive
and terminator sequences, 7-13
internal terminator, 4-66
KEYPAD, 6-116 to 6-117
LIGHTS, 6-118 to 6-119
LIST, 6-120 to 6-123
listing the
with FHELP, 6-80
LOG, 6-124 to 6-125
LOGICAL, 6-126 to 6-128
MAKE_FILE_NAME, 6-130 to 6-133
MERGE, 6-134 to 6-140
MERGE_LINE, 6-141 to 6-142
MERGE_LIST, 6-143 to 6-146
MODE, 6-147 to 6-149
nesting
See FOR, MERGE, SCRIPT
NEWDIR, 6-150 to 6-151
NEWLIB, 6-152 to 6-154
NEXT_LIST, 6-155 to 6-156
OA$CLI_GET_SYMBOL, 6-157 to 6-158
OA$CLI_GET_VALUE, 6-159 to 6-160
OA$CNV_CONVERT, 6-161 to 6-162
OA$CNV_DIST_LIST, 6-163 to 6-164

Functions (Cont.)
OA$CNV_FROM_SCROLL, 6-167, 6-224 to 6-227
OA$CNV_MERGE, 6-165 to 6-166
OA$CNV_TO_SCROLL, 6-168, 6-224 to 6-227
OA$DSA_CACHE_STATUS, 6-169 to 6-170
OA$ENT_SET_xxx, 6-171 to 6-172
OA$FBT_INSTALL_LIBRARY, 6-173 to 6-174
OA$FBT_LIST_LIBTREE, 6-175 to 6-176
OA$FBT_LIST_TREE, 6-177 to 6-178
OA$FBT_REMOVE_LIBRARY, 6-179 to 6-180
OA$FBT_WRITE_LIBRARY, 6-181 to 6-182
OA$FLD_NEXT_LIST, 6-188
OA$FLD_nnn, 6-183 to 6-187
See also individual functions
OA$FLD_RECOGNITION, 6-189
OA$FLD_SCROLL_OFF, 6-190
OA$FLD_SEARCH, 6-191
OA$FLO_CLEAR_VA, 6-192
OA$FLO_CLOSE_LIB, 6-193 to 6-194
OA$FLO_OPEN_LIB, 6-195 to 6-196
OA$FRM_SET_FIELD, 6-197
OA$HAR_CHECK_DECCRT, 6-198 to 6-199
OA$INI_GLOBALS, 6-201
OA$INI_INITIALIZE, 6-202 to 6-203
OA$INI_LOGICALS, 6-204
OA$INI_OPEN_LIB, 6-205
OA$INI_SET_DEFAULT_DIR, 6-206
OA$INI_SUBPROCESS, 6-207 to 6-208

Functions (Cont.)
OA$KEY_COMMAND, 6-209
OA$LIST_xxx, 6-210 to 6-211
OA$LOG_DISPLAY, 6-212 to 6-213
OA$LOG_TO_FILE, 6-214 to 6-215
OA$MENU_LEVEL_PUSH, 6-216 to 6-217
OA$MENU_SHOW_STACK, 6-218 to 6-219
OA$MRG_TTOUT, 6-220 to 6-221
OA$MSG_PURGE, 6-222
OA$MSG_TRAP, 6-223
OA$SCL_nnn, 6-224 to 6-227
See also individual functions
OA$SCP_STACK_DUMP, 6-228
OA$STAT_GET_STATISTICS, 6-229 to 6-230
OA$SUB_CLOSE, 6-231
OA$SUB_OPEN, 6-232
OA$SYM_CLOSE, 6-233
OA$SYM_GET_SYMBOL, 6-234 to 6-235
OA$SYM_MAKE_SYMBOL, 6-236 to 6-237
OA$SYM_OPEN, 6-238 to 6-239
OA$TRA_OUT, 6-240
OA$TRA_SET, 6-241 to 6-243
OA$TRM_BRDCST_INIT, 6-244 to 6-244.1
OA$TRM_BRDCST_STOP, 6-244.2 to 6-244.3
OA$TRM_INIT, 6-245
OA$TRM_INQUIRE, 6-246
OA$TRM_PORT_LINE, 6-247
OA$TRM_PORT_LOG_ON, 6-248
OA$TRM_PORT_OFF, 6-249
OA$TRM_PORT_ON, 6-250
OA$TRM_PORT_PRINT, 6-251

Functions (Cont.)
OA$TXL_CLOSE, 6-252
OA$TXL_COMPILE, 6-253
OA$TXL_DIRECTORY, 6-254
OA$TXL_INSTALL, 6-255
OA$TXL_LIST, 6-256
OA$TXL_OPEN, 6-257 to 6-258
OA$TXL_REMOVE, 6-259
OA$VAL_SET_VALID, 6-260 to 6-261
PARSE_USER, 6-262 to 6-263
pipelining, 6-9
source function, 6-9
PORT_OFF, 6-264
PORT_ON, 6-264
PRINT_SCREEN, 6-265 to 6-266
PROMPT, 6-267 to 6-270
PROMPT_ID, 6-271 to 6-272
PURGE_FILE, 6-273 to 6-276
PUT_FIELD, 6-277 to 6-278
QUEUE_BATCH, 6-279 to 6-283
QUEUE_PRINT, 6-284 to 6-288
RENAME, 6-289 to 6-290
REPEAT, 6-291 to 6-292
SAVE, 6-293 to 6-295
SCRIPT, 6-296 to 6-297
SET_DEFAULT, 6-298 to 6-299
SET_MENU, 6-300 to 6-301
SHOW_FORM, 6-302
SORT_LIST, 6-303 to 6-305
SPAWN, 6-306 to 6-307
stacking, 6-4 to 6-9
See also Function lists
STATISTICS, 6-308 to 6-309
TEST_SPEC, 6-310 to 6-311
TIME, 6-312 to 6-313
trapping errors from, 6-6
types of, 6-1
UNWIND, 6-314 to 6-315
user-defined, 6-10 to 6-11

Functions (Cont.)
restriction on implementing, 6-10
calling standards, 16-22
VAX/VMS requirements, 16-21
VERSION, 6-316 to 6-317
WAIT, 6-318 to 6-320
WPSPLUS$CBI_CLOSE_WINDOW, 6-321
WPSPLUS$CBI_CLOSE_WINDOW2,6-322
WPSPLUS$CBI_OPEN_WINDOW, 6-323 to 6-324
WPSPLUS$CBI_OPEN_WINDOW2, 6-325 to 6-326
WPSPLUS_SPELL, 6-327
WRITE, 6-328 to 6-329
YESNO_PROMPT, 6-330 to 6-332
YESNO_PROMPT_ID, 6-333 to 6-334

G

Generic menu design, 5-1 to 5-8
/GET form qualifier, 3-14
and /GET_SAVE field qualifier, 3-15
processed after /PRE_FUNCTION, 3-14
GET function, 6-101 to 6-107, 16-17
and complex DSRs, 6-107
from the subprocess, 7-11, 7-12
GET OA$DCL, 7-19
GET OA$FUNCTION
special GET destination and SAVE, 6-294
/GET_SAVE field qualifier, 4-16 to 4-18
See also /GET
and /GET form qualifier, 3-16
Gold BACKSPACE
previous page in a form set, 3-80

Gold TAB
next page in a form set, 3-80
Gold-key sequences
and Recognition, 4-46
in a scrolled region, 4-34
in menu choice field, 3-31
predefined, 4-56 to 4-63

H
Hard Copy Mode, 3-16, B-1 to B-5
See also MODE function
providing information while in, 4-18
/HARD field qualifier, 4-18 to 4-19
/HARD form qualifier, 3-16
Help
and Gold H, 4-59
facility, 14-1 to 14-22
See also HELP function
additional topics, 14-14 to 14-18
HELP forms, 14-5
keyboard-style help, 14-5
library, 14-6
cross-references, 14-14 to 14-18
line-1 (direct), 14-17
modifying, 14-18
module format (menu), 14-10
module format (nonmenu), 14-6
restrictions, 14-20
switching to another, 14-21
Help form
.TYPE HELP, 3-6
/HELP form qualifier, 3-17, 14-2
HELP function, 6-108 to 6-111
backing off, 14-4
called after DISPLAY, 6-61
called from a prompt, 6-269
context checking, 14-2
and choice-field contents, 14-3
from a prompt, 14-3, 14-22
from the editor, 14-3
Help function
context checking (Cont.)
on a menu, 14-2
on a nonmenu, 14-2
with /HELP, 14-2
from the default editor, 14-5
syntax restrictions, 6-110
/HIGHLIGHT form qualifier, 3-17
performance impact, 3-18
used with FMS field highlighting, 3-18

I
IFEXIT function, 6-112 to 6-113
in function lists, 6-6
INCLUDE function, 6-114 to 6-115, 10-16
Indexed data files
See Data files
Indexed files, RMS
defined by entry forms, 3-37
Initialization and flow control, 2-1 to 2-6
partial, 2-3
standard sequence of, 2-2
Input Processing facility, 3-31
Installation Verification Procedure
See IVP
Interrupt menu
and Gold I, 4-60
/INVALID field qualifier, 4-4, 4-19 to 4-22
and /RECOGNITION, 4-21
on entry form key fields, 3-46
Iterative symbol substitution
in FOR-function calls, 6-90

K
/KEY entry form qualifier, 3-50
and /BEGIN, 3-51
and /DATA, 3-48, 3-50
and /FIELDS, 3-49, 3-51
and /SAVE_START, 3-52
highlighting by, 3-51

Key of reference, RMS
in an FDL-created file, 3-39
Key, NEXT LIST
and /LIST field qualifier, 4-60
keypad ., 4-60
Keyboard directives
See also individual directives
recognized at prompts, 6-269
where they are recognized, 4-63
Keyboard keys
default definitions of, 4-56 to 4-63
redefined by /SCROLL, 4-34
Keyboard, defining/redefining the, 4-63 to 4-65
/KEYPAD form qualifier, 3-18
KEYPAD function, 6-116 to 6-117
Keypad keys
See also Terminator key(s)
defining/redefining, 4-63 to 4-65
0 — EXIT SCREEN key, 4-59, 6-113
— NEXT LIST key, 4-60
reserved keys, 4-65
/KEYS entry form qualifier
ignored during file creation, 3-38
Keys, data file
ALL-IN-1 versus RMS, 3-39
specified with FDL, 3-39

L

Layered products
ACMS, 6-13 to 6-15
DATATRIEVE, 6-65
Left Brace character
used as script mnemonic delimiter, 15-3
Libraries, form, 3-2, 16-9
opening/reopening
with NEWLIB, 6-152 to 6-154
LIGHTS function, 6-118 to 6-119
/LIST field qualifier, 4-22 to 4-23
and /LIST select form qualifier, 3-70
List form
.TYPE LIST, 3-6
LIST function, 6-120 to 6-123
as a pipeline source, 6-10
LIST key
Gold L, 4-57
List processing, 10-21 to 10-27
See also MERGE, MERGE_LIST
with MERGE, 6-134 to 6-140
with MERGE_LINE, 6-141 to 6-142
WPS-PLUS
MERGE_LIST, 6-143 to 6-146
SORT_LIST, 6-303 to 6-305
TEST_SPEC, 6-310 to 6-311
/LIST select form qualifier, 3-69
and /LIST field qualifier, 3-70
and SEL_STYLE subfunction, 3-70
Local symbol
table, 27
Local symbols, 8-4
/LOG form qualifier, 3-19
LOG function, 6-124 to 6-125
LOGICAL function, 6-126 to 6-128
from the subprocess, 7-12
Logical name, 27
See also Logicals
Logicals, 7-16 to 7-18
main process
and LOGICAL function, 6-126 to 6-128
main process versus subprocess, 7-17

M

/MAIL form qualifier, 3-19
MAIL function, 6-129
See also Electronic Messaging
subfunctions, 11-10 to 11-51
ANSWER, 11-11
ATTACH, 11-13
AUTO_FORWARD, 11-13
AUTO_REPLY, 11-14

MAIL function (Cont.)
CANCEL, 11-15
CANCEL_AUTO_REPLY, 11-15
CC, 11-15
CLOSE_MESSAGE, 11-16
CREATE, 11-16.1
DEFAULT_MESSAGE, 11-19
DEFER, 11-20
DELETE, 11-20
DELIVERY_RECEIPT, 11-21
DETACH, 11-22
DIST_COPY, 11-22
DIST_CREATE, 11-23
DIST_DELETE, 11-23
DIST_EDIT, 11-24
DIST_LIST, 11-24
EDIT, 11-25
EXCHANGE_CURMES, 11-26
EXPAND_DIST_LIST, 11-26
FILE_ATTACHMENT, 11-27
FILE_BODY, 11-28
FILE_MESSAGE, 11-29
FORWARD, 11-30
FORWARDABLE, 11-32
FUNCTION, 11-32
GET, 11-33
INDEX, 11-33
MAIL_DOCUMENT, 11-34
NEXT, 11-36
NO_DELIVERY_RECEIPT, 11-37
NON_FORWARDABLE, 11-37
ORIGINAL, 11-37
OUTBOX, 11-38
POP_CURMES, 11-38
PRINT, 11-39
PRIORITY, 11-40
PUSH_CURMES, 11-41
MAIL function (Cont.)
READ, 11-42
READ_RECEIPT, 11-43
SEND, 11-44
SHOW, 11-46
STATUS, 11-46
SUBJECT, 11-47
TEXT, 11-48
TO, 11-48
VAXMAIL_CHECK, 11-49
Mailboxes, 7-3
and the subprocess, 7-2
use of in applications, 16-20
MAKE_FILE_NAME function, 6-130 to 6-133
from the subprocess, 7-11
Mathematical calculations
operators and operands for, 6-38
with COMPUTE, 6-37 to 6-41
with the Desk Calculator form, 3-58
Mathematical computations
setting decimal places for, 4-11, 6-54 to 6-55
with /COMPUTE field qualifier, 4-7
with Gold #, 4-58
Menu form, 3-24 to 3-36
choice field processing, 3-29
creating a complex, 3-33 to 3-36
Named Data, 3-25
processing
flow control, 3-24
qualifiers
/CAPTIVE, 3-26
/CHOICE (required), 3-25
/FIELDS, 3-26
/MORE, 3-27
/ONCE, 3-28
required features, 3-24
selecting the current item, 4-58, 4-62
selecting the first item, 4-62
.TYPE MENU, 3-6

Menu form processing, 3-28
complex menus, 3-33 to 3-36
Menu levels, 3-36
Menu options, stacking, 3-32
restrictions when, 3-33
Menu stack, prior, 3-35
Menu stacks, 3-36
MERGE function, 6-134 to 6-140
as a pipeline source, 6-10
examples, 10-22 to 10-27
flow control, 10-25, 10-27
header file
and OA$PAGE_LENGTH, 6-136
header-file, 10-26
list file, 10-24
and OA$MERGE_KEY, 6-136
template directives, 6-137 to 6-139
continuation lines, 6-139
MERGE_LINE function, 6-141 to 6-142
template directives with, 6-141
MERGE_LIST function, 6-143 to 6-146
See also SORT_LIST, TEST_SPEC
Message buffer
viewing
with Gold W, 4-60
Message facility, 14-22 to 14-26
See also OAMESS.MSG
and Help, 14-26
message buffer, 14-24
viewing, 14-25
message severity, 14-24
message syntax, 14-23
MESSAGES key
Gold W, 4-60
Mnemonic delimiters
script, 15-3
Mnemonic, scripts, 15-3
/MODE entry form qualifier, 3-44
MODE function, 6-147 to 6-149
Modes
Command (Hard Copy)
providing information while in, 4-18
.MORE, 3-7
.MORE directive, 3-7
/MORE menu qualifier, 3-26, 3-27
in FORM function calls, 3-9
sample application of, 3-34
Multinode features
overview, 12-1
Multipage form
See Form set

N

Named Data, 3-28, 30
dynamically redefining, 3-75
with /RESET, 3-77
modifying, 16-6
reorganizing, 16-9
searching for, 3-7
use of by ALL-IN-1, 3-4
viewing
with Gold N, 4-60
Nesting functions
See FOR, MERGE, SCRIPT
NEWDIR function, 6-150 to 6-151
NEWLIB function, 6-152 to 6-154, 16-11
/NEXT form qualifier, 3-19
and the prior menu stack, 3-75
NEXT LIST key
and /LIST field qualifier, 4-60
keypad ., 4-60
NEXT PAGE key
Gold TAB, NEXT SCREEN, 4-60
NEXT_LIST function, 6-155 to 6-156
from the subprocess, 7-11
$NEXT_LIST_ITEM
maintained with NEXT_LIST, OA$FLD_NEXT_LIST, 7-11

Nonstandard form, 3-80
/NOWRITE entry form qualifier, 3-52

O
&OA
function-call prefix, 6-2, 6-137
OA
function-call prefix, 6-2, 7-5
OA$BUILD:OASDF.BLI
and user-defind functions 6-11
OA$BUILD:HLPSOURCE.HLB
Help-text sources, 14-18
OA$CLI_GET_SYMBOL
function, 6-157 to 6-158
OA$CLI_GET_VALUE
function, 6-159 to 6-160
OA$CNV_CONVERT function, 6-161 to 6-162
OA$CNV_DIST_LIST
function, 6-163 to 6-164
OA$CNV_FROM_SCROLL
function, 6-167, 6-224 to 6-227
See also /SCROLL
OA$CNV_MERGE function, 6-165 to 6-166
OA$CNV_TO_SCROLL
function, 6-168, 6-224 to 6-227
See also /SCROLL
OA$CURDOC
special symbol
and CABINET CLEAR_CURDOC, 9-23
and CABINET COPY, 9-25
and CABINET CREATE, 9-26
and CABINET CROSSFILE, 9-26
and CABINET CURRENT, 9-27
and CABINET DELETE_OR_REFILE, 9-29
and CABINET NEXT_FOLDER, 9-36
and CABINET NEXT_NUMBER, 9-37
and CABINET NEXT_TITLE, 9-40
and CABINET SELECT, 9-45
OA$DATA
and LOGICAL function, 6-128
subprocess versus main process, 7-17
OA$DATA:ATTENDEE.DAT
system attendee file, 13-1
OA$DATA:CALACCESS.DAT
CALENDAR access file, 13-1
OA$DATA:DAF.DAT
See SDAF
OA$DATA:FORMAT.DAT
Format master file, 10-28
OA$DATA:MEETING.DAT
system meeting file, 13-1
OA$DATA:PENDING.DAT, 9-12
and CABINET GET_PENDING, 9-31
and CABINET PUT_PENDING, 9-42
system Pending file, 9-2
OA$DATA:username.A1CAL
the personal calendar file, 13-1
OA$DATE
special symbol
and DATE_CONVERT, 6-48
validation DSR, C-2
OA$DATE_FULL
special symbol
and DATE_CONVERT, 6-48
OA$DATE_NBS
special symbol
and DATE_CONVERT, 6-48
OA$DCL
special GET destination, 6-52, 16-23
OA$DEFAULT_EXTENSION
special symbol, 6-131, 10-3
used to create file name, 4-12
OA$DIR
field-name keywords, C-3
special DSR, C-3

OA$DISPLAY
special GET destination, 6-61
OA$DSA__CACHE__STATUS
function, 6-70, 6-169 to 6-170
OA$EDIT__BUFFER
text data set, 10-10
OA$EDIT__FILE
special symbol, 6-71
OA$EDIT__INI__FILE
special symbol, 6-72
OA$EDIT__PURGE__COUNT
special symbol, 6-72
OA$EDITOR
special symbol, 6-73
OA$ENT__SET__xxx function,
6-171 to 6-172
OA$FBT__INSTALL__LIBRARY
function, 6-173 to 6-174
OA$FBT__LIST__LIBTREE
function, 6-175 to 6-176, 6-194,
6-196
OA$FBT__LIST__TREE
function, 6-177 to 6-178
OA$FBT__REMOVE__LIBRARY
function, 6-179 to 6-180
OA$FBT__WRITE, 16-11
OA$FBT__WRITE__LIBRARY
function, 6-181 to 6-182
OA$FIELD__TERM
special symbol, 7-13
and BACKSPACE, 4-57
and Gold BACKSPACE,
PREV SCREEN, 4-61
and Gold TAB, NEXT
SCREEN, 4-61
and keypad 0, 4-59
and PROMPT, 6-267, 6-271
and RETURN, 4-58
and TAB, 4-62
OA$FLD__BOTTOM function,
4-67, 6-183
and Gold B, 4-57
OA$FLD__DONE function,
4-67, 6-184
and PROMPT, 6-267, 6-271
and RETURN, Gold F, 4-58
OA$FLD__DOWN function, 4-67,
6-184
and Down Arrow, 4-58
OA$FLD__EXIT function,
4-67, 6-184
and IFEXIT, 6-113
and keypad 0, Gold K, Gold
Q, 4-59
and PROMPT, 6-267, 6-271
and the prior form, 3-75
OA$FLD__NEXT function, 6-184
and TAB, 4-62
OA$FLD__NEXT__LIST
function, 4-67, 6-188
and /LIST field qualifier, 4-60
and the NEXT LIST key, 4-60
from the subprocess, 7-11
OA$FLD__nnn functions, 4-66,
6-183 to 6-187
See also individual functions
OA$FLD__PAGE__BACK
function, 4-67, 6-185
and Gold BACKSPACE, PREV
SCREEN, 4-61
OA$FLD__PAGE__BOTTOM
function, 6-185
OA$FLD__PAGE__FORWARD
function, 4-67, 6-185
and Gold TAB, NEXT
SCREEN, 4-60
OA$FLD__PAGE__TOP
function, 6-185
OA$FLD__PREV function, 4-67,
6-185
and BACKSPACE, 4-56
OA$FLD__RECOG function
and Gold L, 4-57
OA$FLD__RECOGNITION
function, 6-189
OA$FLD__SCROLL__OFF
function, 6-190
OA$FLD__SEARCH function,
6-191
and Gold S, 4-57

OA$FLD_TOP function, 4-67
and Gold T, 4-62
OA$FLD_UP function, 4-67, 6-186
and Up Arrow, 4-62
OA$FLO_CLEAR_VA function, 6-192
OA$FLO_CLOSE function
and DUMP_CACHE, 6-70
OA$FLO_CLOSE_LIB function, 6-193 to 6-194
and DUMP_CACHE, 6-194
and NEWLIB, 6-194
OA$FLO_OPEN function, 16-11
and DUMP_CACHE, 6-70
OA$FLO_OPEN_LIB function, 6-195 to 6-196
and DUMP_CACHE, 6-196
and NEWLIB, 6-196
OA$FORM_DISPOSE
special symbol
See also Form dispositions
and CABINET, 6-6
and OA$FLD_BOTTOM, 4-57
and OA$FLD_DONE, 4-58
and OA$FLD_EXIT, 4-59
and OA$FLD_NEXT, 4-62
and OA$FLD_PAGE_BACK, 4-61
and OA$FLD_PAGE_FORWARD, 4-61
and OA$FLD_PREV, 4-57
and OA$FLD_TOP, 4-62
and YESNO_PROMPT, 6-330, 6-334
set by OA$FLD_nnn functions, 4-67
summary of values for, 4-68
OA$FORM_TERMINATOR
special symbol, 7-13
and BACKSPACE, 4-57
and Gold BACKSPACE, PREV SCREEN, 4-61
and Gold TAB, NEXT SCREEN, 4-61
and keypad 0, 4-59
and PROMPT, 6-267, 6-271
and RETURN, 4-58
and TAB, 4-62
and YESNO_PROMPT, 6-330, 6-334
terminator values, 4-69
OA$FRM_SET_FIELD function, 6-197
OA$FUNCTION
special GET destination
and ERROR, 6-77
and LIST, 6-122
and LOG, 6-125
and LOGICAL, 6-128
and MERGE, 6-140
and NEWDIR, 6-151
and PURGE_FILE, 6-275
OA$FUNCTION, GET
special GET destination
and SAVE, 6-294
OA$HAR_CHECK_DECCRT function, 6-198 to 6-199
OA$HELP_BOTTOM
HELP form, 14-5
OA$HELP_TOP
HELP form, 14-5
OA$INI_GLOBALS function, 6-201
OA$INI_INITIALIZE function, 6-202 to 6-203
OA$INI_LOGICALS function, 6-204
OA$INI_OPEN_LIB function, 6-205
OA$INI_SET_DEFAULT_DIR function, 6-206

OA$INI__SUBPROCESS
function, 6-207 to 6-208
OA$KEY__COMMAND
function, 6-209
OA$LIB
and LOGICAL function, 6-128
subprocess versus main process, 7-17
OA$LIB:ACTITEM.FDL, 3-42
used to create OAUSER:ACTITEM.DAT, 3-42
OA$LIB:FINDCOM.COM
and SET__DEFAULT, 6-299
OA$LIB:MEMRES.FLB
memory-resident form library, 3-2, 16-10
OA$LIB:OAFORM.FLB
altering search order of, 3-3
common form library, 3-2
OA$LIB:OAHELP.HLB
See also Help library
ALL-IN-1 help library, 6-110
default help library, 14-7
OA$LIB:STANDARD.FLB
prototype form library, 3-2
OA$LIB:USER.FLB
user's form library, 3-2
OA$LIST__xxx function, 6-210 to 6-211
OA$LOG__DISPLAY function, 6-212 to 6-213
OA$LOG__TO__FILE function, 6-214 to 6-215
OA$MAIDES
validation DSR, C-7
OA$MAIL__ADD__ADDR
special validation DSR, 11-50
OA$MAIL__ADDRESS
special validation DSR, 11-50
validation DSR, C-8
OA$MAIL__COUNT
special symbol
and CABINET GET__PENDING, 9-32
OA$MAIL__NUM__ADDRESSES
MAIL-related logical, 11-9
OA$MAILPRIO
special validation DSR, 11-50
validation DSR, C-8
OA$MENU__LEVEL__PUSH
function, 6-216 to 6-217
OA$MENU__REMAINDER
special GET destination, 3-33
OA$MENU__SHOW__STACK
function, 6-218 to 6-219
OA$MERGE__KEY
special symbol
and MERGE, 6-136
and MERGE list file, 10-25
OA$MERGE__LINE
special GET destination, 10-26
and MERGE__LINE, 6-142
OA$MERGE__REC
See OA$MERGE__KEY
OA$MRG__TTOUT function, 6-220 to 6-221
See also MERGE, MERGE__LINE
and MERGE, 6-135
and MERGE__LINE, 6-142
OA$MSG__FAC
special symbol, 14-25
OA$MSG__ID
special symbol, 14-25
OA$MSG__PURGE function, 6-222
OA$MSG__SEV
special symbol, 14-25
OA$MSG__TEXT
special symbol, 14-25
OA$MSG__TRAP function, 6-223
OA$PAGE__LENGTH
special symbol
and MERGE, 6-136, 10-27
OA$PAGE__NUMBER
special symbol
and MERGE, 6-136, 10-27
OA$PROMPT__HELP
special symbol, 14-3
and %OA-I-LASTLINE, 6-62
and HELP, 6-269
and HELP function, 6-111
and prompts, 6-269

OA$PROMPT_TEXT
special symbol
and PROMPT, 6-267, 6-271
OA$SCL_BOTTOM function
and Gold Down Arrow, 4-34
OA$SCL_DOWN function
and the Down Arrow, 4-34
OA$SCL_FIRST_PAGE function
and Gold T, 4-34
OA$SCL_LAST_PAGE function
and Gold B, 4-34
OA$SCL_NEXT_PAGE function
and Gold TAB, 4-34
OA$SCL_nnn functions, 6-224 to 6-227
See also individual functions
OA$SCL_PRIOR_PAGE function
and Gold BACKSPACE, 4-34
OA$SCL_TOP function
and Gold Up Arrow, 4-34
OA$SCL_UP function
and the Up Arrow, 4-34
OA$SCP_STACK_DUMP function, 6-228
OA$SCRIPT_EXIT_HANDLER
special symbol
and Gold X, 15-4
and Gold X, CTRL/C, 4-59
OA$SCRIPT_PSIB
script pseudo-input buffer, 15-3
and script text lines, 15-3
in script DO Mode, 15-4
OA$SCRIPT_TERM
script pseudo-input buffer terminator, 15-4
OA$SCRIPT_TEXT
script pseudo-input buffer text, 15-4
OA$SEL_ADDR
special symbol
used with FOR function, 3-73
OA$SEL_COUNT
special symbol
used with FOR function, 3-73
OA$SEL_DSAB
special symbol
used with FOR function, 3-73
OA$SEL_KEY
special symbol
and validation post-functions, 4-51
used with FOR function, 3-74
OA$SEL_LINE
special symbol
used with FOR function, 3-74
OA$SHARE_CUR
and CABINET SHARE_DOCUMENT, 9-47
OA$SHARExxx logicals, 9-19
See also File Cabinet shared area
OA$SRC_COUNT
special symbol
used with FOR function, 3-74
OA$STAT_GET_STATISTICS function, 6-229 to 6-230
OA$STATUS
special symbol
and CABINET, 6-6
and CABINET NEXT_FOLDER, 9-36
and CABINET subfunction calls, 9-15
OA$SUB_CLOSE function, 6-231
OA$SUB_OPEN function, 6-232
OA$SYM_CLOSE function, 6-233
OA$SYM_GET_SYMBOL function, 6-234 to 6-235
OA$SYM_MAKE_SYMBOL function, 6-236 to 6-237
OA$SYM_OPEN function, 6-238 to 6-239
OA$TABLE
validation DSR, C-8
OA$TRA_OUT function, 6-240
OA$TRA_SET function, 6-241 to 6-243
OA$TRM_BRDCST_INIT function, 6-244 to 6-244.1

OA$TRM_BRDCST_STOP
function, 6-244.2 to 6-244.3
OA$TRM_INIT function, 6-245
OA$TRM_INQUIRE function, 6-246
OA$TRM_PORT_LINE
function, 6-247
OA$TRM_PORT_LOG_ON
function, 6-248
OA$TRM_PORT_OFF
function, 6-249
OA$TRM_PORT_ON function, 6-250
OA$TRM_PORT_PRINT
function, 6-251
OA$TXL_CLOSE function, 6-252
OA$TXL_COMPILE function, 6-253
OA$TXL_DIRECTORY
function, 6-254
OA$TXL_INSTALL function, 6-255
OA$TXL_LIST function, 6-256
OA$TXL_OPEN function, 6-257 to 6-258
OA$TXL_REMOVE function, 6-259
OA$TXL_SEARCH_LAST, 15-6
OA$VAL_SET_VALID
function, 4-50, 4-54, 6-260 to 6-261
OA$VIEW_FIELDS
View Fields form, 4-63, 6-149, 16-18
OA$VIEW_MESSAGES
and Gold H, 4-60
and message buffer, 14-25
View Messages form, 4-60, 16-18
OA$VIEW_ND
View Named Data form, 4-60, 6-149, 16-18
OA$YESNO
validation DSR, C-9
OA$YN
validation DSR, C-9
OAHELP
logical name, 14-21
main-process logical, 6-110
OAINI.COM
BYE function redefined in, 6-24
EXIT function in, 6-79
subprocess log-in file, 7-2
OAMAILBOX
and DCLMAILBOX, 7-4
calling multiple functions through, 6-8
mailbox from subprocess, 7-4
multiple writes to, 7-7
syntax while writing to, 7-6
OAMESS.MSG
message file, 14-23
updating the, 14-23
OAUSER
and COMMAND function, 6-36
and DCL function, 6-53
and SET_DEFAULT, 6-299
subprocess versus main process, 7-17
OAUSER:ACTITEM.DAT
file structure, 3-40
OAUSER:CALACCESS.DAT
Action Items file, 13-1
OAUSER:DAF.DAT
See PDAF
OAUSER:DOCDB.DAT
personal File Cabinet index, 9-2, 9-4
keys to, 9-4, 9-5
reorganizing
and CABINET REORG_DOCDB, 9-44
OAUSER:PASTE.TMP
See Scratch Pad
/ONCE form qualifier
in entry forms, 3-52
/ONCE menu qualifier, 3-28
/ONE_ENTRY entry form qualifier, 3-53
and /SAVE_START, 3-53
/OPTIONAL field qualifier, 4-21, 4-23 to 4-24, 4-49

/OVERLAY form qualifier, 3-20

P

Pad, scratch
  and Gold *, 4-61
PARSE_USER function, 6-262 to 6-263
  from the subprocess, 7-11
PDAF, 9-2, 9-9, 9-46
  and OAUSER:DOCDB.DAT, 9-9
Performance
  and /HIGHLIGHT, 3-18
  and FMS field highlights, 16-23
  and RMS tuning, 16-23
  and symbol types, 8-15, 16-23
  command procedures versus scripts, 16-22
  effect of stacking functions on, 6-8
Permanent symbol, 34
Permanent symbol table, 34
  username.PST, 8-31
Permanent symbols, 8-2, 8-31 to 8-36
Personal Document Attributes File
  See PDAF
Pipeline mechanism, 6-9
  source functions
    MERGE, 6-134, 6-141
  with COPY, 6-44
PORT_OFF function, 6-264
PORT_ON function, 6-264
  and PRINT_SCREEN, 4-61
/POST_FUNCTION
  appended in FORM function calls, 3-9
/POST_FUNCTION field qualifier, 4-24 to 4-25
  and the prior menu stack, 3-75
  versus /POST_FUNCTION form qualifier, 4-24
/POST_FUNCTION form qualifier, 3-20, 3-21
  and the prior menu stack, 3-75
/PRE_FUNCTION
  appended in FORM function calls, 3-9
/PRE_FUNCTION field qualifier, 4-25 to 4-26
  and the prior menu stack, 3-75
/PRE_FUNCTION form qualifier, 3-21
  and the prior menu stack, 3-75
Precompiling forms, 16-11
  with the PCF option, 16-11
PREVIOUS PAGE key
  Gold BACKSPACE, PREV SCREEN, 4-61
PRINT SCREEN key
  and PRINT_SCREEN function, 4-61
  Gold P, 4-61
PRINT_SCREEN
  saving in PASTE.TMP, 4-61
PRINT_SCREEN function, 6-265 to 6-266
  from the editor, 4-61
  Gold P, 4-61
Printing files
  See QUEUE_PRINT
Prior menu stack, 3-35
Profile
  See User Profile
Profile, user
  privileges
    PRVCMD, 3-23, 3-32
    PRVDCL, 3-32
    PRVSUP, 3-23
Programming Functions, 16-12 to 16-14
/PROMPT field qualifier, 4-26 to 4-27
PROMPT function, 6-267 to 6-270
  and OA$FLD_nnn functions, 4-67
  and user-defined keys, 6-269
  from the subprocess, 7-13

PROMPT_ID function, 6-271 to 6-272
and OA$FLD_nnn functions, 4-67
from the subprocess, 7-13
PRVCMD
User Profile privilege, 3-23, 3-32, 6-2
PRVDCL
User Profile privilege, 3-32, 4-58
PRVSUP
User Profile privilege, 3-23
PURGE_FILE function, 6-273 to 6-276
PUT_FIELD function, 6-277 to 6-278
/PUT_SAVE field qualifier, 4-27 to 4-28

Q

Qualifiers
See also Field qualifiers, Form qualifiers
Field versus Form, 3-10
Qualifiers, field
used to define a field, 4-2
QUEUE_BATCH function, 6-279 to 6-283
versus COMMAND, 6-282
QUEUE_PRINT function, 6-284 to 6-288
Quit form
See EXIT SCREEN

R

Recognition
full
and Gold S, 4-57
partial
and Gold L, 4-57
with /INVALID, 4-19
with /RECOGNITION, 4-28
with /VALID, 4-47
/RECOGNITION field qualifier, 4-28 to 4-30
with /VALID, 11-50
Record collection
controlling output of, 4-55 to 4-56
with /USE_FORM, 4-55
creating
with FOR function, 6-83 to 6-91
with select form, 3-68, 3-71
with validation qualifiers, 4-55
displaying
with SEL_DISPLAY subfunction, 3-72, 6-87
formatting
with /SHOW, 4-45
with /STYLE select-form qualifier, 3-70
with search form .SOUT directive, 3-67
with SEL_STYLE subfunction, 3-72, 6-87
number of records in
OA$SEL_COUNT, 3-73
putting unique records into
with /UNIQUE, 4-47
refining
with FOR FIRST, 6-84
with FOR UNIQUE, 6-84
saving
with /LIST select-form qualifier, 3-69
selecting from
with /STYLE select-form qualifier, 3-70
with OA$SEL_ADDR, 3-73
with OA$SEL_KEY, 3-74
with OA$SEL_LINE, 3-74
with SEL_CHOICE subfunction, 3-72, 6-87
source of records in a
OA$SEL_DSAB, 3-73
viewing
with Gold L, 4-57
with Gold S, 4-57

Record Management Services
See RMS
Record selection expression
See RSE
Redefining the keyboard, 4-63 to 4-65
Relational operators, 6-85
RENAME function, 6-289 to 6-290
REPEAT function, 6-291 to 6-292
in function lists, 6-7
/RESET form qualifier, 3-22
in FORM function calls, 3-9
using the, 3-77
RESULT
common DCL symbol, 7-12
and LOGICAL, 6-126
updating, 6-102
DCL symbol
set by OAFLD_DONE, OAFLD_EXIT, 4-67
Retrieving symbol values
with /GET_SAVE, 4-16
RETURN key
and OA$FLD_DONE, 4-58
Right Brace character
used as script mnemonic delimiter, 15-3
RMS indexed files
defined by entry forms, 3-37
RMS key of reference
See also Data file keys
in an FDL-created file, 3-39
RMS tuning
for improved performance, 16-23
RSE
differences between search and select forms, 3-61
in field validation, 4-30 to 4-32
in search form .TYPE entry, 3-63
in the FOR function, 6-83
/RSE_INVALID field qualifier, 4-4, 4-30 to 4-32
/RSE_RECOG field qualifier, 4-30 to 4-32
/RSE_VALID field qualifier, 4-4, 4-30 to 4-32

S

SAVE function, 6-293 to 6-295
from the subprocess, 7-11
/SAVE_START entry form qualifier, 3-53
and /KEY key fields, 3-53
and /ONE_ENTRY, 3-53
Scratch Pad
and Gold*, 4-61
SCRIPT function, 6-296 to 6-297
See also DO
Script processing
uses of scripts, 15-1
Scripts, 15-1 to 15-34
See also SCRIPT, DO functions
adding comments to, 15-25
block output
beginning, 15-10
calling functions from, 15-14, 15-15
conditional processing in, 15-17
controlling cursor location in, 15-12, 15-13, 15-26
controlling screen appearance in, 15-11, 15-12, 15-24, 15-26, 15-30
debugging, 15-29
directives
.ABORT, 15-4, 15-9
as functions, 15-31
.BLOCK, 15-10
.CHECK_FIELD, 15-10
.CHECK_FORM, 15-11
.CLEAR, 15-11
.CUR_KEEP, 15-12
.CUR_RESTORE, 15-13
.CUR_SAVE, 15-13
.CURSOR, 15-12
.DISPLAY, 15-13
.EXIT, 15-4, 15-14
.FUNCTION, 15-14

Scripts (Cont.)
.FX, 15-15
.FZ, 15-15
.GET_FIELD, 15-16
.GOTO, 15-17
.IF, 15-17
.JUDGE, 15-19
.LABEL, 15-21
.PAUSE, 15-21
.PROCESS, 15-22
and OA$SCRIPT_PSIB, 15-3
.PROMPT, 15-24, 15-28
.REFRESH, 15-24
.REMARKS, 15-25
.RESET_CURSOR, 15-26
.SET_PAUSE, 15-26
.SET_REFRESH, 15-26
.SHOW_FORM, 15-27
.SINX, 15-28
.TEXT, 15-29
.TRACE, 15-29
.TRACE_ON, 16-15
.VIDEO_ATT, 15-30
.WAIT, 15-30
displaying forms in, 15-27
example, 15-32
exiting from, 4-59, 15-4, 15-9, 15-14
field processing in, 15-16
flow control in, 15-17, 15-19, 15-21
format, 15-7
graphics in, 15-13
interactive directives
and OA$SCRIPT_PSIB, 15-3
mnemonic delimiters, 15-3
mnemonics for input, D-1
nested, 15-2
output from, 15-29
pausing in, 15-21, 15-26
processing input while in, 15-22
prompting from, 15-24, 15-28, 15-30
synchronizing, 15-10, 15-11
text lines, 15-8
terminators in, 15-8
tracing
with Gold (, 6-34
two modes of, 15-2
DO, 15-2
mixing the, 15-2
Scripts mnemonic, 15-3
Scripts, command procedures
versus performance in applications, 16-22
/SCROLL field qualifier, 4-33 to 4-44, 6-224
Scrolled region
creating with FMS, 4-34
identifying with /SCROLL, 4-33
navigating in
with OA$SCL_nnn functions, 6-224
Scrolling
display-only, 4-43
Scrolling functions
See OA$SCL_nnn
SDAF, 9-2, 9-9, 9-46
and OAUSER:DOCDB.DAT, 9-9
Search form, 3-60 to 3-68
and FOR function, 3-65
Named Data, 3-61
.SFILE directive, 3-65
.SOUT directive, 3-67
processing, 3-68
qualifiers, 3-65
.TYPE entry syntax, 3-60, 3-63 to 3-65
differences between select and search, 3-61
signaling errors in, 3-65
.TYPE SEARCH, 3-6
SEARCH key
Gold S, 4-57
SEARCH132 arg form
and record collection output, 3-67

Searching text files
with search form .SFILE directive, 3-65
SEARCHKEY arg form
and record collection output, 3-67
Section
globally shared, 16-11
Segmented form
See Form set
SEL_CHOICE
FOR subfunction
and OA$SEL_LINE special symbol, 3-74
in select forms, 3-72
special FOR subfunction, 6-87
SEL_DISPLAY
FOR subfunction
and /SHOW, 4-45
in select forms, 3-72
special FOR subfunction, 6-87
SEL_STYLE
FOR subfunction
and /LIST, 3-70
in select forms, 3-72
special FOR subfunction, 6-87
Select form, 3-68 to 3-74
Named Data, 3-69
processing, 3-72
summary, 3-74
.TYPE line, 3-71
qualifiers, 3-69
/LIST, 3-69
/STYLE, 3-70
required features, 3-69
.TYPE SELECT, 3-6
Selection List
and NEXT_LIST, 6-155 to 6-156
created by /LIST select-form qualifier, 3-70
created by search form, 3-60
identifying
with /LIST field qualifier, 4-22
identifying next selection in, 4-23
opening and reading
with OA$FLD_NEXT_LIST, 4-60
Sequential files
See Text files
SET_DEFAULT function, 6-298 to 6-299
and COMMAND function, 6-36
and DCL function, 6-53
and NEWDIR, 6-150
and OAUSER, 6-299
SET_MENU function, 3-32, 6-300 to 6-301
.SFILE directive
in search form Named Data, 3-65
syntax, 3-66
Shared Document Attributes File
See SDAF
Shared libraries
in the File Cabinet, 9-19 to 9-20
/SHOW field qualifier, 4-45 to 4-46, 4-55
and SEL_DISPLAY, 4-45
with multiple data sets, 4-45
SHOW_FORM function, 3-4, 3-12, 6-302
and nonstandard forms, 3-80
SORT_LIST function, 6-303 to 6-305
See also MERGE_LIST, TEST_SPEC
.SOUT directive
in search form Named Data, 3-67
syntax, 3-67
SPAWN function, 6-306 to 6-307
and the primary subprocess, 7-19
Special symbol
table, 41

Special symbols, 8-3, 8-16 to 8-30
DTR$, 6-66
performance versus DSRs, 16-23
read-only, 8-15 to 8-30
read-write, 8-15 to 8-30
STAT$, 6-308
used with FOR function, 3-73
write-only, 8-15 to 8-30
Special-function keys
See Keyboard directives
Stacking functions, 6-4 to 6-9
See also Function lists
Stacking menu options, 3-32
restrictions when, 3-33
Statistics
ALL-IN-1 release
and VERSION, 6-316
data set
and OA$DSA_CACHE_STATUS, 6-169 to 6-170
form library
and OA$FBT_LIST_LIBTREE, 6-175 to 6-176
and OA$FBT_LIST_TREE, 6-177 to 6-178
process
and STATISTICS, 6-308 to 6-309
STATISTICS function, 6-308 to 6-309
/STORE field qualifier, 4-46
/STYLE select form qualifier
and /LIST, 3-71
and SEL_STYLE subfunction, 3-72
used to format record collection, 3-70
Subprocess, 7-1 to 7-23
accessing with DCL function, 6-51 to 6-53
calling function lists from, 6-8
creating an alternate
with SPAWN, 6-306
initializing, 7-1
obtaining user input while in, 7-20 to 7-23
passing data to programs in, 7-20
receiving data from through OAMAILBOX, 7-4
running command procedures in, 7-19
sending data to through DCLMAILBOX, 7-4
sending text files to with INCLUDE, 6-115
trapping errors while in with ERROR, 6-6, 6-76 to 6-77
Subsystem
defined, 1-6
Subsystems, 3-24
interaction of, 1-6
/SUPER form qualifier, 3-23
Support, terminal, B-2
Symbol table
local, 8-4
permanent, 8-2
opened during initialization, 2-3
Symbols, 8-1 to 8-36
basic types, 8-2 to 8-5
DSRs, 8-4
field name, 8-2
local, 8-4
permanent, 8-2
quoted string, 8-2
special, 8-3
concatenating in lists, 8-11
creating, 8-14
with /COPY field qualifier, 4-8
dates and times in, 8-13
See also Date/time formats
defining and using
in the subprocess, 7-8 to 7-16

Symbols (Cont.)
getting values from, 8-15
with /GET_SAVE, 4-16
with GET, 6-101 to 6-107
in function calls, 6-12
in the main process, 8-15
in the subprocess, 8-15
iterative substitution, 8-12
performance differences, 8-15
putting values into, 8-14
with /PUT_SAVE, 4-27
substrings, 8-10
values stored in multiple, 8-15
zero-suppressing numeric, 8-11
Symbols, DCL
creating
with GET, 6-103
SYS$INPUT
mailbox to subprocess, 7-4

T
TAB
and OA$FLD_NEXT, 4-62
Terminal Modes
See Modes
Terminal Support, B-2, B-5
TERMINATOR
common DCL symbol, 7-13
and interactive functions, 7-13
DCL symbol
and PROMPT, 6-267, 6-271
set by OAFLD_DONE, OAFLD_EXIT, 4-67
Terminator functions
See OA$FLD_nnn
internal, 4-66
Terminator key(s)
processing, 4-66 to 4-69
in menu choice field, 3-31
TEST_SPEC function, 6-310 to 6-311
See also MERGE_LIST, SORT_LIST
Text data sets, 10-2
categories
ASCII, 10-2
COMPOUND, 10-2
DX, 10-2
DXT, 10-2
MAIL_ATTACHMENT, 10-2
MAIL_HEAD, 10-2
PIPELINE, 10-3
WPL, 10-3
OA$EDIT_BUFFER, 6-75
writing to
with DESTINATION, 6-58
Text files
copying
with COPY, 6-43 to 6-44
creating
with EDIT, 6-71 to 6-73, 10-3
with MERGE, 6-134 to 6-140
with MERGE_LINE, 6-141 to 6-142
with MERGE_LIST, 6-143 to 6-146
deleting, 6-56 to 6-57
displaying
See also OA$CNV_FROM_SCROLL
See also OA$CNV_TO_SCROLL
with LIST, 6-120
extensions and editors, 10-3
formatting, 10-28 to 10-33
including text into
with INCLUDE, 6-114 to 6-115
internal structure versus editor style, 10-21
modifying
See also OA$CNV_FROM_SCROLL
See also OA$CNV_TO_SCROLL
with EDIT, 6-71 to 6-73
(default editor), 10-4 to 10-19

Text files (Cont.)
naming
with /DOC, 4-11
printing
with QUEUE_PRINT
on an attached printer, 6-264
processing, 10-1 to 10-33
searching
with search form .SFILE
directive, 3-65
validating existence of
with /FILE_CHECK, 4-14
WPS-PLUS
and MERGE_LIST, 6-143
to 6-146
formatting
with FORMAT, 6-97 to
6-100
Text processing, 10-1 to 10-33
editor menu, 6-74 to 6-75
Time
formatting
See Dates and times
TIME function, 6-312 to 6-313
Time Management, 13-1 to 13-30
activities, 13-2
CALENDAR, 6-26
organization, 13-1
/TITLE form qualifier, 3-23
TRACE keys, VERIFY and
Gold (,Gold), 4-62
TXL '
and the OAGBL.BLI
module, 15-5
building and modifying, 15-5
search order, 15-6
changing with
OA$TXL_SEARCH_LAST,
15-6
to increase performance, 15-5
.TYPE or type
See Form .TYPE

U

/UNIQUE field qualifier, 4-47, 4-52
UNWIND function, 6-314 to 6-315
Up Arrow
and OA$FLD_UP, 4-62
calls OA$SCL_UP in scrolled
region, 4-34
/USE_FORM field qualifier, 4-47
/USE_FORM validation
subqualifier, 4-21, 4-49, 4-51, 4-55
with /INVALID, 4-19
with /RECOGNITION, 4-28
with /RSE_nnn, 4-31
/USER form qualifier, 3-24
User Profile
ACTITEM field, 6-18
and NEWLIB, 6-152 to 6-154
DOCFRM field
and NEWDIR, 6-150
Form Libraries field (FRMLIB)
use of, 3-3
FRMLIB field, 16-9
opened during initialization, 2-2
privileges
PRVCMD, 3-23, 3-32, 6-2
PRVDCL, 3-32, 4-58
PRVSUP, 3-23
User-defined functions
See Functions, user-defined
User-defined keys
See Keyboard directives

V

/VALID field qualifier, 4-4, 4-47
to 4-50
and /RECOGNITION, 4-49
on entry form key fields, 3-45
with /RECOGNITION, 11-50
Validation
and OA$VAL_SET_VALID,
6-260
field qualifiers, 4-4
/AUTO, 4-5
combining, 4-54
/FILE_CHECK, 4-14 to 4-16
/INVALID, 4-19 to 4-22
/OPTIONAL, 4-23 to 4-24

Validation
field qualifiers (Cont.)
/RECOGNITION, 4-28 to 4-30
/RSE_INVALID, 4-30 to 4-32
/RSE_RECOG, 4-30 to 4-32
/RSE_VALID, 4-30 to 4-32
/UNIQUE, 4-47
/VALID, 4-47 to 4-50
optional
with /OPTIONAL, 4-23
with /RECOGNITION, 4-28
through field qualifiers, 4-51 to 4-56
with RSEs, 4-30 to 4-32
Validation expression, 4-51, 45
DSR in a, 4-20, 4-29, 4-48, 4-51
with /INVALID, 4-19
with /RECOGNITION, 4-28
with /VALID, 4-47
VAX Information Architecture
and ALL-IN-1, 1-4
overview, 1-3
VERIFY and TRACE keys
Gold (,Gold), 4-62
Verifying
command procedures
with Gold (, 6-34
Version 2.X, converting to
form libraries, 16-9
VERSION function, 6-316 to 6-317
View form, 16-18
See also individual forms
.TYPE VIEW, 3-6
VMS
file names
See File names

W

WAIT function, 6-270, 6-318 to 6-320
and OA$FLD_nnn functions, 4-67
from the subprocess, 7-13
WPINDX
select form
processing of, 3-72
WPS-PLUS
and MERGE_LIST, 6-143 to 6-146
and SORT_LIST, 6-303 to 6-305
and TEST_SPEC, 6-310 to 6-311
WPSPLUS$CBI_CLOSE_WINDOW, 6-321
WPSPLUS$CBI_CLOSE_WINDOW2 function, 6-322
WPSPLUS$CBI_OPEN_WINDOW function, 6-323 to 6-324
WPSPLUS$CBI_OPEN_WINDOW2 function, 6-325 to 6-326
WPSPLUS_SPELL function, 6-327
WRITE function, 6-328 to 6-329
and entry-form modes, 3-78
called by FOR, 6-91

Y

YESNO_PROMPT function, 6-270, 6-330 to 6-332
See also PROMPT
and OA$FLD_nnn functions, 4-67
from the subprocess, 7-13
YESNO_PROMPT_ID function, 6-333 to 6-334
See also PROMPT
and OA$FLD_nnn functions, 4-67
from the subprocess, 7-13

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA and Puerto Rico call **800–258–1710**

In Canada call **800–267–6146**

In New Hampshire, Alaska or Hawaii call **603–884–6660**

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: P&SG Business Manager

INTERNATIONAL

DIGITAL EQUIPMENT CORPORATION
P&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

*Any prepaid order from Puerto Rico must be placed with the Local Digital Subsidiary: 809–754–7575

Reader's Comments

Your comments and suggestions are welcome. They will help us in our continuous effort to improve the quality and usefulness of our publications.

1 How would you rate this manual for:

	good	fair	poor
completeness of information	_____	_____	_____
accuracy of information	_____	_____	_____
ease of use (clarity, organization)	_____	_____	_____

2 Did you find errors in this manual? Please specify by page and paragraph, or mark corrections on pages and attach copies to this form.

	page	paragraph
a) Incorrect information:	_____	_____
b) Information left out:	_____	_____
c) Hard to understand:	_____	_____

3 What suggestions do you have for improving this manual?

_____

_____

4 What parts of the manual were especially good at helping you understand things?

_____

_____

5 Please check the boxes that apply to you.

- ☐ Technical
- ☐ Nontechnical
- ☐ Management
- ☐ Nonmanagement
- ☐ Other (explain)_____
- ☐ Used other word processors
- ☐ Used computers
- ☐ Use the word processing unit yourself

Name_____ Title_____

Company_____

Street_____ City_____

State_____ Zip_____ Date_____

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: Manager OFFICE PROGRAM ZK02-1/N20
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, N.H. 03062

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line

AD-DC93A-T1

ALL-IN-1

OFFICE MENU

Application Programmer's Reference Volume 2: Functions

Update Package #1, January 1986

This package updates the ALL-IN-1 Office Menu Application Programmer's Reference Volume 2 (Order No AA-DC93A-TE) for ALL-IN-1 Version 2.1.

In this package, a bar (▌) in the outside margin shows where information has been changed or added. A bullet (●) shows where information has been deleted. These marks do not appear in the Table of Contents or Index.

Insert the pages as follows.

Insert page	In place of
Guide to ALL-IN-1 Information	Guide to ALL-IN-1 Information
iii – xlii	iii – xlii
xlvii – l	xlvii – l
6-11 – 6-12	6-11 – 6-12
6-26.1 – 6-26.2	(new pages)
6-29 – 6-30	6-29 – 6-30
6-43 – 6-44	6-43 – 6-44

(continued)

Insert page	In place of
6-53 – 6-58	6-53 – 6-58
6-83 – 6-86.1	6-83 – 6-86
6-129 – 6-132	6-129 – 6-132
6-135 – 6-136	6-135 – 6-136
6-149 – 6-152	6-149 – 6-149
6-199 – 6-200	6-199 – 6-200
6-243 – 6-246	6-243 – 6-246
6-261 – 6-262	6-261 – 6-262
6-273 – 6-274	6-273 – 6-274
6-285 – 6-288	6-285 – 6-288
6-327 – 6-330	6-327 – 6-330
Index-1 – 36	Index-1 – 38

AD-DC93A-T2

OFFICE MENU

Application Programmer's Reference Volume 2: Functions

Update Package #2, April 1987

Update the Application Programmer's Reference Volume 2 (AA-DC93A-TE) with this package. You must first have updated the APR with update package #1, January 1986 (AD-DC93A-T1). When you have inserted both packages, the APR will be updated for ALL-IN-1 Version 2.2.

In this package, a bar (▌) in the outside margin shows where information has been changed or added. A bullet (●) shows where information has been deleted. These marks do not appear in the Table of Contents or Index.

Insert the pages as follows.

Insert page	In place of
xxi – xxii	xxi – xxii
6-19 – 6-20	6-19 – 6-20
6-26.1 – 6-26.2	6-26.1 – 6-26.2
6-67 – 6-68	6-67 – 6-68
6-133 – 6-134	6-133 – 6-134
6-137 – 6-138	6-137 – 6-138
6-227 – 6-228	6-227 – 6-228

(continued)

Insert page	In place of
6-244.1 – 6-244.3	6-244.1 – 6-244.3
6-255 – 6-256	6-255 – 6-256
6-267 – 6-268	6-267 – 6-268
6-327 – 6-328	6-327 – 6-328
Index-15 – 16	Index-15 – 16
Index-21 – 22	Index-21 – 22
Index-25 – 28	Index-25 – 28

AA-DC93A-TE
Printed in U.S.A.